TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kristjan Narusk 185722

AS SEB PANK EESTI PABERIVABA PROJEKTI EDASIARENDUS

Bakalaureusetöö

Juhendaja:
Tõnn Talpsepp
Doktorikraad

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kristjan Narusk

13.05.2021

Annotatsioon

Käesoleva lõputöö käigus jätkati AS SEB Panga paberivaba teeninduse äriprojekti arendust. Antud projekt on kliendi-telleri vaheliseks töö hõlbustamiseks mõeldud lahendus, mille käigus on võimalik kas tahvelarvuti või kaugnõustamise teel pakkuda pangatooteid klientidele. Meeskonnaprojekti ning lõputöö raames võeti eesmärgiks lõpetada Paberivaba Panga projekti reklaamihalduse osa, asendada aegunud hostnime päringu lahendus ning lisada andmete peegeldamise funktsioon paberivaba projekti. Projekti arendus käis kahes osas – esialgu arendati lõpuni reklaamihaldus, seejärel liiguti lõputöö raames edasi andmete peegeldamise ning eelnevalt eksisteerinud hostinime lahenduse asendamise juurde.

Tulemusena said töökorras enamus ette antud nõuetest täidetud. Reklaamihalduse poolelt sai lõpule viidud administraatori paneeli vajalikud muudatused, millest kõige tähtsamaks oli reklaami salvestamine. Esialgse plaani järgi oleva ID-kaardi autentimise arenduseni ei jõutud eelnevate probleemide lahendamisest tingitud ajaliste viivituste tõttu. Arendamise jooksul proovis meeskond järgida kõiki agiilse metoodikaga seotud viise ning põhimõtteid. Meeskonda kuulus tootejuht, IT meeskonna juht, arhitekt, testija ning 2 juuniorarendajat.

Lõputöö hulka kuulub 18 joonist ning 54 lehekülge.

Abstract

Continuation of the Paperfree Bank solution development at AS SEB Bank

The following bachelor thesis is based on the development continuation of the Paperfree Project at AS SEB Bank. Skandinaviska Enskilda Banken AB, also known as SEB is a Swedish financial institution, established in 1852 by André Oscar Wallenberg. SEB has now grown out to be more than a banking group, providing clients all over the world with world-class financial services. The current Paperfree project is developed for the clients and users in the Baltics, with the main solution and testing done in Estonia.

This project is a solution for facilitating work between a customer and a teller, which can be used to offer banking products to customers either via a tablet computer or by remote advisory. The goal of the team project and the thesis was to complete the advertising management part of Paperfree Bank solution, to replace the expired host name querying solution and to further develop the data mirroring function between bank teller's portal and Paperfree Bank. The development of the project took place in two parts – the first part was to finish the development steps of commercial management application. The second part was mirroring client data and replacing the old hostname querying solution.

As a result, most of the requirements were met in working order. On the commercial management side, the changes to the admin panel were completed, the most important of which was saving the commercial file with data. According to the initial plan, the development of the existing ID card authentication could not be solved due to time delays. During development, the team followed all the methods and keywords associated with the agile methodology. The team consisted of one product owner, one IT team lead, an architect, a tester and 2 junior developers.

The following thesis is written in Estonian and contains 54 pages and 18 schematics.

Sisukord

Jooniste loetelu

Lühendid ja mõisted

NBSF	Tellerite töökeskkond AS SEB Pangas
Back-end	Loogika pool, mis vastutab andmete salvestamise ja manipuleerimise eest
Front-end	Andmete visualiseerimise ning kuvamise pool
Java	Objektorienteeritud programmeerimiskeel, üldjuhul kasutatud *back-end* arenduses
Pipeline (äri)	Progressi kirjeldus, mis on jaotatud etappideks, et saavutada tähtsam eesmärk
Hostname	Nimetus, mis on määratud arvutivõrguga ühendatud seadmele
HTML	Veebilehtede märgenduskeel, kasutatakse struktuuri loomiseks
OpenID	Lihtne välisrakendus, mille abil on kasutajaid võimalik verifitseerida
MP4	Digitaalse multimeediummahuti formaat, kasutatakse video ja heli salvestamiseks
Maven	Projekti manageerimise tööriist, peamiselt kasutatakse Java puhul
UX	Ingliskeelsest sõnast *User Experience* ehk kasutajakogemus
UI	Inglisekeelsest sõnast *User Interface* ehk kasutajaliides, vastutab andmete kuvamise ja lehekülje navigatsiooni eest

1 Sissejuhatus

AS SEB Panga äriarenduse initsiatiivil otsustati 2019. aasta lõpus välja hakata arendama uut Paberivaba Panga lahenduse projekti. Hetkel eksisteerib Paberivaba Panga lahendusest töötav versioon, mis on ajale jalgu jäämas. Seda just selle tõttu, et uusi funktsionaalsusi on keeruline lisada, hoolduskulud on kõrged, raamistikuversioonid on iganenud ning arhitektuurilised lahendused on tarkvara arenduse mõttes vananenud. Kogu projekti, nii uue kui ka vana, eesmärk on parandada panga jätkusuutlikku kuvandit ning tuua esile vähenenud paberikasutus igapäevaste toimingute tegemisel. Uus lahendus baseerub AS SEB Panga telleritele mõeldud NBSF süsteemil, mille kaudu päritakse nii kliendi kui ka telleri andmeid. Arenduses osales mitmeliikmeline arendusmeeskond, mille koosseis muutus projekti elutsükli käigus mitmel korral. Arendajatest kuulus meeskonda peamiselt 1 vanemarendaja ning 2 juuniorarendajat, kellest üks on antud bakalaureusetöö autor, Kristjan Narusk. Arendusmeeskonna juhi rollis on Aleksei Nikokošev, kellest sai projekti vältel üks lõputöö juhendajatest. Lõputöö on puhtalt kirjutatud läbi tudengi teostatud arenduse ja analüüsi kaudu. Antud lahenduse lõplik valmimisaeg on veel teadmata, kuid projekt plaanitakse klientideni tuua juba 2021. aasta teise kvartali lõpuks. Kõigi nõuete realiseerimiseni on veel mahukas tööhulk jäänud, mistõttu on kvartalipõhiste etappide ajapiirangud nihkunud edasi. Bakalaureusetöö on kirjutatud Tallinna Tehnikaülikooli Äriinfotehnoloogia eriala meeskonnaprojekti aine jätkuks, mille raames valmis paberivaba projekti reklaamihalduse rakenduse pool. Nüüdseks teostati paberivaba projekti telleri arvuti *hostname*-i päring ning andmete peegeldamise NBSF-ist. *Hostname*-i päringu tulemusena saadi eemaldada eelnev lahendus, mis töötas Chrome veebibrauseri laienduse põhjal. Lisaks oli plaanis lõplikult teostada paberivaba keskkonda ID-kaardiga autentimislahendus, kuid millest sai valmis ainult uurimustegevuslik osa eelnevate eesmärkide arendusega seotud komplikatsioonide tõttu.

Meeskonnaprojekti aine raames sai valmis rakenduse reklaamihalduse pool, mille abil on telleril võimalik automatiseerida tahvelarvutitel jooksvaid reklaamide kestvust ning vahetust. Edasine arendus hõlmas endas ekraanide ehk *front-end* poolel asuvate infomallide loomist HTML markeeringukeeles, info pärimist NBSF tellerkeskkonnast, *hostname* päringu arendust NBSF keskkonnas ning ID-kaardiga seotud autentimise uurimist. Seetõttu võetigi lõputöö eesmärkideks eelmainitud nõuete teostamine lõpule viia, millest ka antud lõputöös lähemalt räägitud on.

Projektil on suur hulk funktsionaalseid nõudeid, millest põhilisemad ning lõputööga seostuvad on järgnevad:

- Klient saab läbi Paberivaba Panga lahenduse kuvada SEB Panga kodulehte, siseneda nii era- kui ärikliendi internetipanka, digiportaali ja näha pangaga sõlmitud lepingu informatsiooni;
- Vajutades tahvli ekraanile kuvatakse kliendile mitu valikut. Üks nendest on SEB Panga koduleht, kus temal on võimalik sisse logida kasutades olemasolevate autentimismeetodeid;
- Telleril on võimalik lahti võtta SEB Panga koduleht või internetipanga rakendus NBSF-ist, kus kasutajal on võimalik sisse logida kasutades olemasolevaid autentimismeetodeid;
- Tahvelarvuti ekraanile puudutades näeb klient mitut valikut. Üks nendest on sisselogimine Eesti Vabariigis kehtiva ID-kaardi, Mobiil-ID või Smart-ID abil Paberivaba Panga keskkonda, kus tal on võimalus siseneda Digiportaali või hakata nägema pooleliolevaid lepinguid, mida teller parasjagu NBSF keskkonnast „peegeledab“;
- Kui tahvelarvuti on ooterežiimis, peab tahvelarvuti näitama reklaame. Nende kuvamist saab katkestada, vajutades tahvelarvuti ekraanile või nõustajapoolse sessiooni loomisega tahvelarvutiga.

Mittefunktsionaalsetest nõuetest võib välja tuua näiteks jõudluse poole andmete „peegeldamise“, reklaamihalduse administraatori paneeli ja *back-end*-i osas. Viimase puhul on näiteks tähtis, et päringud oleks lihtsad ning kiiresti töötavad. Ühilduvuse poolelt on tähtsateks nõueteks näiteks see, et Paberivaba Panga projekt töötaks viimasel Google Chrome veebilehitseja verisoonil. Lisaks peaks antud projekt töötama kõigi ettevõttesiseste turvalahendustega. Turvanõuete poolest on tähtsal kohal kindlasti punkt, mis kirjeldab, et Paberivaba Panga lahendus ei tohiks kliendi andmeid mingil määral salvestada. Muuhulgas peaks klient saama ligi ainult enda andmetele. Lisaks peaksid tellerid saama antud lahenduse administraatori keskkonda kasutada vaid juhul, kui nad on autoriseeritud läbi OpenID. Töökindluse aspektist lähtudes peaks Paberivaba Panga lahendus olema Eesti, Läti ja Leedu klientidele saadaval ~98% lahtiolekuajast igal kuul. Lisaks ei tohiks hooldustöödele kuluda üle 5% tööajast. Tahvelarvutitele on samuti sätestatud mitmeid nõudeid nagu näiteks legaalse

Windows operatsioonisüsteemi, antiviiruse tarkvara ning Bitlocker krüpteermistarkvara olemasolu.

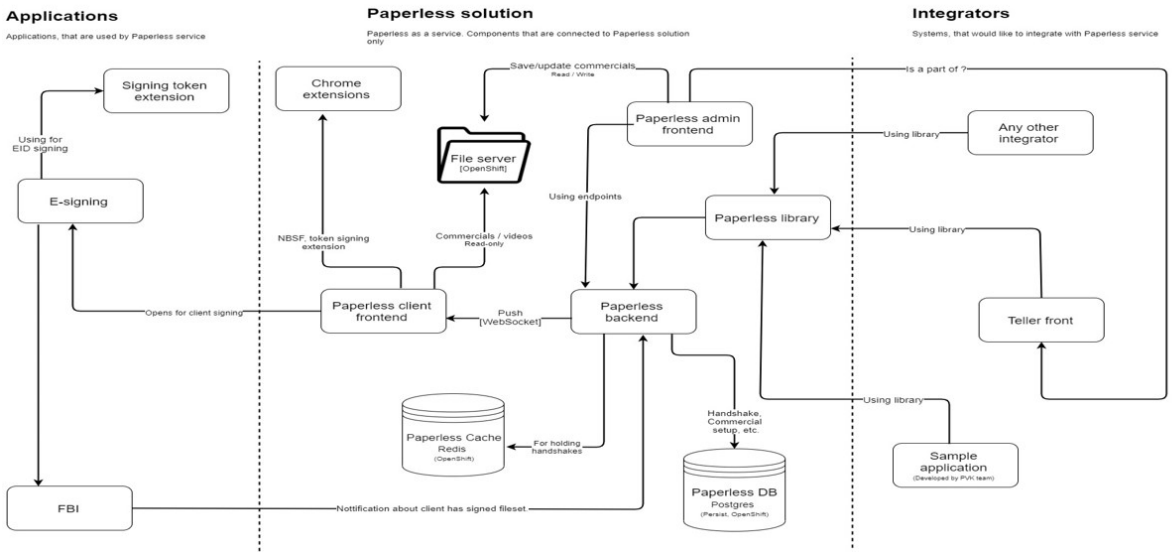
Antud bakalaureusetöös on detailselt kirjeldatud ülevaade tehtud tööst ning analüüsist. Lõputöö koosneb viiest osast: sissejuhatus, metoodika, töö tulemused, analüüs ja järeldused ning kokkuvõte. Lisades on välja toodud autori panus arendustegevuses ning eneseanalüüs.

Metoodika all on lahti kirjutatud tööprotsessi, tööriistade ning objekti detailne kirjeldus, mis hõlmab endas ülevaadet kogu protsessist. Töö tulemuste all aga kõik teostatud nõuded ning valmissaadud rakendus koos muu tööga. Analüüsis on välja toodud tehtud töö tulemuste põhjused ning valideerimine – miks selliseid nõudeid, tööriistu ja koodi on kasutatud. Lisades on detailselt välja toodud, kuidas jagunes töö arendajate vahel ning kuidas analüüsib autor enda hinnangul projekti üldiselt.

2 Metoodika

2.1 Objekti detailne kirjeldus

Täiendavalt on vaja antud projekti AS SEB Pangal mitmel põhjusel. Esiteks on klientide nõuded ärivajaduste mõttes suurenenud märkimisväärselt, mis tõttu on vanem versioon paberivaba lahendusest ajale jalgu jäämas. Uusi funktsioone juurde arendada oli äärmiselt keeruline oma vananenud arhitektuurilise poole tõttu. Lisaks oli uue lahenduse üheks nõudeks väiksemad hoolduskulud nii ajalises kui ka rahalises mõttes. Uue lahenduse meeskonna visiooniks sai muuta paberivaba lahendus üheks põhiliseks tööriistaks kliendiga töötades ja anda neile veel lihtsam ja tõhusam viis pangateenuste kasutamiseks. Projekti arhitektuur sai ümber töötatud ning valminud lahendus joonistati välja skeemina (Joonis 1).

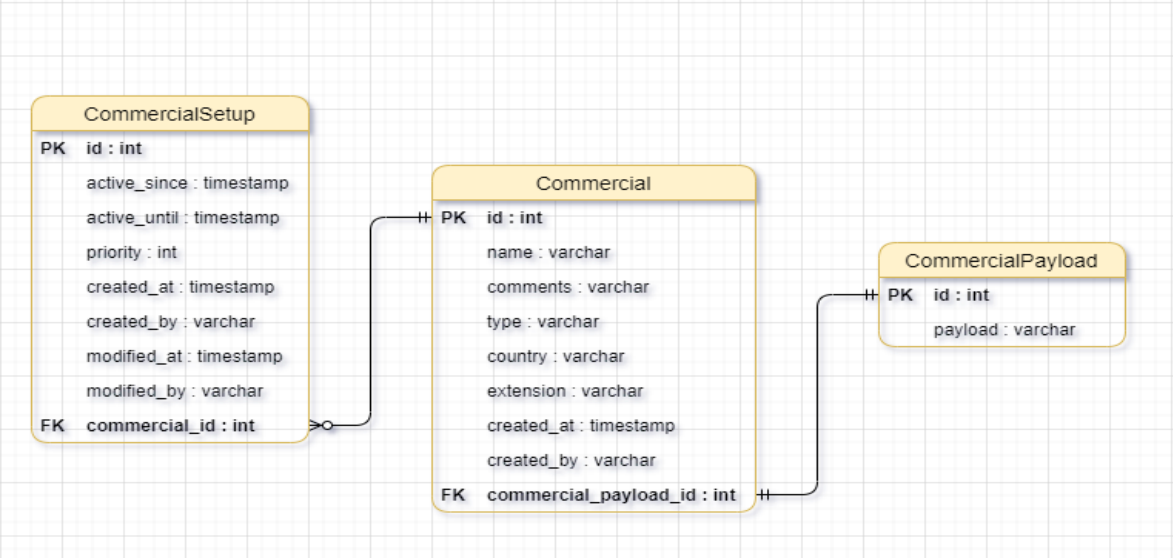


Joonis 1. Paberivaba projekti top-level arhitektuuri skeem

Antud meeskonnaprojekti esimeseks eesmärgiks oli lõpule viia reklaamihalduse administraatoripaneeli arendus, seejärel sai edasi liigutud andmete peegeldamise juurde NBSF süsteemist Paberivaba Panka. Lõpuks kaotati ära Chrome-i veebibrauseripõhine esmane *hostname*-i päringu lahendus, mis asendati NBSF süsteemi *back-end* koodimuudatustega. Reklaamihalduse poolel, millel algselt oli hinnanguliselt valmis pool *back-end* ja *front-end* lahendusest, sai esiteks lisatud reklaami sisestamise vorm, millele saab kasutaja ligi „Lisa reklaam“ nupu alt. Tegu on 4-sammuliuse vormiga, mille täitmise lõpus kuvatakse kasutajale sisestatud andmetest koosnev ankeet ning vajutades „Salvesta“ salvestatakse uus reklaam andmebaasi koos sisestatud täiendavate andmetega. Lisaks uue reklaami salvestamisele sai

lisatud riigipõhine reklaamide listi sorteerimine, detailvaade koos seadistustega ning mõlema, nii reklaami andmete kui seadistuste muutmine.

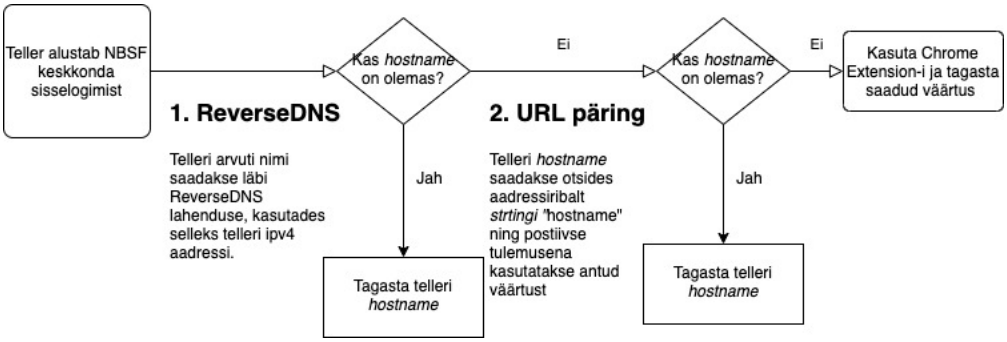
Enne projektiga tegelema asumist oli olemas põhi, millele asuti juurde arendama vajalikke nõudeid. Adminpaneeli puhul oli olemas juba eelnevalt mainitud *back-end* ja *front-end* osa. Projekti *back-end* pool põhineb Spring Boot raamistikul ning mille programmeerimiskeeleks on Java. Lisaks oli kogu Paberivaba Panga projekti puhul varasemalt üles seadistatud Postgre andmebaas (Joonis 2).



Joonis 2. Paberivaba projekti reklaamihalduse andmebaasi struktuur

Reklaamihaldusest edasi liiguti Paberivaba Panga projekti andmete peegeldamise osa juurde. Ka siinkohal oli olemas mingisugune *front-end* põhi, millele tuli esialgselt juurde lisada uued lehekülje mallid nii era- kui ärikliendi jaoks. Antud kliendile mõeldud *front-end* poolel oli varasemalt olemas reklaami näitamiseks mõeldud komponent, valikute komponent ning väike osa kliendi potentsiaalsetest lepingutest. Järgmiseks sai tehtud suur hulk uusi kaustasid projekti *front-end components* kausta, kuhu omakorda hakati välja arendama malle, mis pidid vastama UX disaineri poolt välja joonistatud *sketch*-idele. Antud malli arendamine käis *trial and error* stiilis, kus pilves olev *sketch* võeti ühele ekraanile lahti ning teisel ekraanil prooviti võimalikult täpselt antud pilti jäljendada, kirjutades selle jaoks HTML koodis asi ümber. Lähtudes puhta koodi põhimõtetest, prooviti võimalikult palju komponente, nagu näiteks reklaamteatised, lehekülje päis ning jalus, taaskasutada, kirjutades need eraldi failidesse. Paralleelselt mallide arendusele lisati *Typescript*-i kasutades funktsionaalsus n-ö. kuulata *back-end* poolt kliendi andmete pärimiseks ning need hiljem antud info mallidele kanda.

Lisaks eelnevale sai lõputöö raames välja arendatud uuemat sorti telleri arvuti *hostname*-i pärimise lahendus, kasutades selleks kolmeastmelist süsteemi (Joonis 3.).



Joonis 3. Telleri arvuti hostname päringu skeem

Antud lahenduse jaoks oli vaja NBSF süsteemis teha muudatused sisselogimise jaoks mõeldud klassi, kus oli vaja kirjutada juurde loogika telleri IP aadressi pärimiseks, selle edasi suunamiseks ReverseDNS klassi ning teise sammuna kasutatava URL päringu kood. Siinkohal tuleb välja tuua ReverseDNS-i loogika varasem olemasolu, kuid selle kasutamata jätmine puhtalt selle tõttu, et ei saadud aru miks antud lahendus ei töötanud. Sellepärast eelnes koodi kirjutamisele eelnevalt pikemat sorti uurimisfaas, kus oli vaja välja selgitada mis tõttu antud klass kastutust ei leia ning kuidas selle probleemiga edasi tuleks minna. Kolmanda sammuna kasutatava Chrome brauseripõhise liidese kasutamine jäeti testimise jaoks esialgselt alles tagavaralahendusena ning eemaldatakse tulevikus kui minnakse projektiga lõplikult *live*-i ehk klientideni.

2.2 Tööriistade kirjeldus

Paberivaba Panga projekti puhul on kasutusel enda *back-end* ning andmebaas. *Back-end* lahendus baseerub Java programmeerimiskeelel ning sarnaselt NBSF-iga, kasutab Spring Boot raamistikku. Algselt oli kasutusel Java versioon 11.0.5 kuid mis uuendati antud aasta alguses versioonile 14. Et Java projekte lihtsam arendada oleks, võeti ka siinkohal kasutusele konfiguratsioonifail nimega Maven. Maven on tööriist, mida saab kasutada mis tahes Java-põhise projekti koostamiseks ja haldamiseks. Arendajatel on võimalik sisehaldussüsteemist alla laadida ning kasutusele võtta arenduskeskkonnad Eclipse ja/või IntelliJ IDEA. Antud valikust eelistati selgelt rohkemal määral IntelliJ keskkonda oma lihtsasti kasutatava kasutajaliidese tõttu.

Paberivaba Panga *front-end* puhul kasutatakse JavaScript programmeerimiskeelel põhinevat raamistikku nimega Angular ning versiooniks oli 6.12.1. Arenduskeskkonna puhul oli samuti jäetud arendajatele valikuruumi – võimaluseks valida kas Visual Studio Code või JetBrain WebStorm. NBSF-i jaoks oli *front-end* kirjutatud *back-end* lahendusega kokku, kasutades siinkohal JSP lahendust. JSP ehk *Jakarta Server Pages* on programmeerimisel kasutatav tehnoloogia/raamistik, mille puhul on võimalik siduda lihtsalt HTML markeeringukeel, javascript ning Java ühes ja samas failis.

Koodi hoiustamiseks kasutati AS SEB Pangas Git versioonihaldurit ning pideva integratsiooni ja edastamise jaoks rakendatakse Jenkins tarkvara. Jenkins on pideva integreerimisega ja avatud lähtekoodiga automatiseerimisserver, mis aitab automatiseerida tarkvara arendamise protsessi osa, mis arendajast enam ei sõltu (C.R. Reidolf, T. Tammaru, L. Väli, 2020). Jenkins töötab sarnaselt teistele populaarsetele pidev integratsiooni tööriistadele nagu näiteks GitLab-i sisse ehitatud omale, tehes peale koodi üleslaadimist rakendusele *build* protsess ehk kompileerimine ning läbides kõik ettenähtud testimiseks mõeldud protsessid. Enne koodi edasiminekut *live* staatusesse, tehti koodile *review* ehk ülevaade. Koodi ülevaateid tehti kusjuures teise arendaja poolt, seda selleks, et nii-öelda puhta vaatega inimene näeks võib-olla seda, mis peamise arendaja jaoks kahe silma vahele jäänud. Antud projekti meeskonnas lasid mõlemad nooremarendajad üksteise koodile ülevaateid teha.

Projekti haldamine ning meeskonnatöö ülevaate jaoks kasutati JIRA-t. Jira on veebis paiknev rakendus, kus on võimalik projektiga seotud ülesanded jagada väiksemateks tükkideks, mida on võimalik hiljem, näiteks sprindi planeerimisel jaotada ära meeskonnaliikmete vahel. Jira kasutamine oli üks igapäeva ülesannetest – hommikuti sai iga meeskonnaliige näha kiiresti ning mugavalt sprindi ülevaadet, mitu päeva on sprindi lõpuni aega, mis on vaja veel teha ning mis on juba valmis. Veel lisaks on Jiras käepäraselt saadaval projekti *backlog* (projektiga seotud tulevased ülesanded, mis sprinti ei jõudunud), väljalasked, aruanded, komponendid ja testid. Ülesannete täitmise ja koodi kirjutamise jaoks kasutati oma git-i haru ehk *branch-i*, mis hiljem integreeriti põhiharuga, kui testimine ja koodi ülevaade korras oli. Koodi integreerimiseks kasutatakse SEB siseselt välja töötatud integratsioonialgoritmi, mis saadab Jira-sse teavituse kui integreerimisel tõrkeid ei esinenud. Mõistagi toimib teavitamine ka vastupidise sündmuse korral. Koodiintegraatori jaoks oli vaja läbi Git-i üleslaetud koodi sõnumisse/kirjeldusse lisada Jira probleemikood –

kolmest tähesümbolist ning numbritest genereeritud kood, mis tähistas ära ülesande, millega üleslaetud kood seotud oli.

2.3 Tööprotsessi kirjeldus

Projekti arendamine algas vajalike arenduskeskkondade ja õiguste tellimisega ning samuti uute keelte ja raamistike kohanemisega. Õnneks ei olnud õppimisfaas pikk, sest varasem kogemus teiste programmeerimiskeeltega aitas märgatavalt. Algse ootuse juures, et programmeerimiskeelelt C# üleminek Java peale oleks konarlik ja pikaleveniv, osutus vastupidiseks. Kindlasti aitas siinjuures kaasa juhendaja tugi. Kohanemise käigus tehti arendajatele selgeks uued nõuded ning milline on lõppeesmärk. Lõputöö arenduse käigus muutusid osad nõuded, näiteks ReverseDNS-i puhul leiti, et antud ülesande puhul saaks lihtsa vaevaga lisada uue, URL päringu lüli tagavara variandina. Lisaks otsustati, et mõistlik oleks Chrome liidese päringu kood esialgu alles jätta ning eemaldada see järk-järgult testimisfaasi käigus.

Paljudes arendusmeetodites, eriti plaanipõhistes, algab töö täieliku nõuete kogumise ja dokumenteerimisega. Alates 1990. aastate keskpaigast leidsid paljud, et see esialgne nõuete dokumentatsiooni samm on pettumust valmistav ja võib-olla võimatu (L. Williams, A. Cockburn, 2003). Selles tulenevalt on suurem osa tarkvaraarendusest migreerunud agiilsele metoodikale. Siit tulenevalt arenduse efektiivsuse tõstmiseks kasutatakse SEB Pangas Scrum metoodikat, mis on üks agiilselt töötava meeskonna alustalasid. Scrum on kergekaaluline agiilne projektijuhtimise raamistik, mida saab kasutada igat liiki iteratiivsete ja inkrementaalsete projektide haldamiseks. Scrum metoodika juures jaotakse arendusperiood ära lühemateks faasideks ehk sprintideks. Sprintidel jaotati ülesandeid kasutades selleks *story point* süsteemi. Iga arendaja sai enda sprindi töömahule vastavalt tööülesandeid ning igat ülesannet hinnati vastavalt kollektiivi hinnangule *story-pointidega*. *Story Point* oma olemuselt on lihtsasti panduna numbriline arv, mis suureneb vastavalt keerukusele Fibonacci arvudena. Pangas kestis üks sprint 2 nädalat.

Sprintide juurde käisid ka *Stand-Up* koosolekud, mis toimusid igapäevaselt kell 11.00 hommikul läbi Microsoft Teams-i vahendusel. *Stand-Up* koosolek kestis tihtilugu 15min, kuid oli päevi, mil kestis kauem ning oli ka päevi, millal ei olnud tiimiliikmetel nii palju jagada, mis tähendas, et koosolek võidi lõppenuks lugeda isegi 5 minutiga. Stand-Up koosolekutel vastasime peamiselt küsimustele nagu „Mida teen täna?“, „Mida tegin eile?“

ning „Kas mul on probleeme, millega tiimikaaslased saaksid aidata?“. Vahel pikenes koosolek kahe meeskonnaliikme vahel, kui tekkinud küsimused või probleemid teisi ei puudutanud. Algselt toimus kogu suhtlus koosolekute ajal inglise keeles, isegi kui arutavad teemad ei puudutanud teistes keeltes kõnelevaid inimesi. Inglise keele vajalikkus oli ühe Leedu haru meeskonnaliikme tõttu, kes 2021 aasta alguses lahkus tiimist. See tõi koheselt kaasa kõikidel koosolekutel keelekasutuse muudatuse eesti keele peale.

Lõputöö arenduse algus tõi endaga kaasa hulga uurimuslikku tööd. See tähendab, et enne arendust tuli kaardistada ning paljuski uurida miks mõni komponent ei tööta või miks on mõni muu lahendus selle asemel tööle võetud. Chrome laienduse eemaldamiseks pidi algselt uurima kas antud ReverseDNS klass töötab. Selle jaoks sai *mock* andmetega, mille alla käis nii enda arvuti kui teiste töötajate IPV4 aadressid, testitud klassi funktsioonide töötamine. Antud funktsioonidesse said IP aadressid sisestatud parameetritena ning tagastatud vastust kontrollitud läbi Windows operatsioonisüsteemi Command Prompt „hostname“ käsurea. Järgnevalt tuli leida lahendus kuidas saada kätte kasutaja IP aadress võttes samal ajal arvesse kõikki sätestatud turva- ja äripoolseid nõudeid.

Tiimivaheline suhtlus toimus enamasti Skype For Business kaudu. Igasugused koosolekud ja failivahetus oli siiski läbi Teams-i. Õnneks on Skype ja Teams tarkvaraliselt väga hästi üksteisega integreeritud, näiteks saab mugavalt liituda Teams koosolekutega läbi Skype-i „Meetings“ valiku. Kõik koosolekud olid suures osas töösisesed ning hõlmasid endas projektiteemalist arutelu, kuid tiimi motivatsiooni tõstmiseks ja üksteisega tuttavaks saamiseks toimus projekti vältel 2 nii-öelda koosviibimist läbi Teams-i.

3 Töö tulemused

Kogu projekti arendus põhines peamiselt agiilsel metoodikal, kus kasutati scrum raamistikku. Seetõttu olid paljud etteantud nõuded ning eesmärgid arendusfaasi vältel muutuvad, mis muutis pideva suhtluse eriti oluliseks. Meeskonnaprojekti esimese poole peamine eesmärk oli selge: arendada lõpuni reklaamihalduse süsteem ning testida läbi vajalikud komponendid. Arenduse käigus andis peamised juhtnöörid ning nõuded ette projekti arhitekt Sergei Pojev ning arendajateks olid Daniil Petuhhov ja Kristjan Narusk. Hiljem, projekti teises faasis käis peamine suhtlus ning edendamine Aleksei Nikokošev-i ja Kristjan Narusk-i vahel.

3.1 Väljatöötatud nõuded

Iga tarkvara projekti arendus algab selgesõnaliste ning konkreetsete nõuete olemasoluga. Siinkohal oli antud projekti nõuded kajastatud JIRA „*Requirements*“ menüüs. Reklaamihalduse näitel oli üheks ülesandeks luua reklaamfaili üleslaadimine ehk Paberivaba Panga *back-end*-i salvestamine. Selleks tuli projekti puhul luua nii *back-end* kui ka *front-end* osade juurde uued nõuded:

Paberivaba Panga projekti *back-end*:

- Luua uued *endpoint*-id*,* et aktsepteerida *payload* aplikatsiooni frondi poolelt. Need *endpoint*-id salvestavad uue reklaamfaili andmebaasi. Videofail peab olema *base64* kodeeringuga, mida saab kasutada failide salvestamiseks andmebaasis.
- Luua kõik eespool nimetatud *endpoint*-ide jaoks vajalikud teenus- ja valideerimisfunktsioonid.
- Luua *endpoint,* mis tagastab tõese või vale vastuse, kui peale viimast front poole küsimist on saadaval uus, värskendatud videofail.

Admin-UI nõuded:

- Luua uus menüüpunkt failide üleslaadimiseks.
- Luua uus vorm reklaami ja faili salvestamiseks. Salvestamisel peaks küsima reklaami nime, riiki ja kommentaare antud faili kohta.
- Luua vorm reklaami seadistuse salvestamiseks. Salvestamisel peaks küsima prioriteeti ja aktiivsuse perioodi.
- Lisada väljade kontrollimine täitmise käigus.

- Enne *back-end* poolele saatmist tuleks fail teisendada *base64* kodeeringusse.

Paberivaba UI ehk kliendi front nõuded:

- Küsi reklaamifaili Paberivaba Panga *back-end* osalt.
- Kuna päritud fail on *base64* kodeeringus, tuleb see dekodeerida ümber mängitavaks failiks, soovitatavalt mp4.
- Luua protsess/funktsioon, mis küsib *back-end* poolelt iga 3 tunni tagant, kas andmebaasis eksisteerib värskendatud reklaamfail. Kui eksisteerib, pärida antud fail ning alustada selle videofaili kuvamist.
- Juhul kui *back-end* poolega puudub ühendus, näidata mõnda eelnevalt salvestatud pilti, et klientidele ei kuvataks tühja ega musta ekraanpilti.
- Taotleda OIDC-lt luba Paberivaba Panga UI rakenduse jaoks.

Chrome veebilehitseja liidese eemaldamiseks ning läbi IP aadressi telleri arvuti *hostname* pärimiseks oli vaja samuti viia sisse muudatusi. Antud ülesande tegi problemaatiliseks eelnev teadmatus süsteemidevahelisest suhtlusest ning varasemast arendusest. NBSF poole nõuded olid järgnevad:

- Kontrollida NBSF *ReverseDNS* klassi õigsust ning töötamist. Vajadusel viia sisse koodi muudatused, et antud funktsioon töötaks korrektselt.
- Luua NBSF *front* poolele funktsioon pärida kliendi ehk kasutaja IPV4 aadress.
- Luua funktsioon edastada eelmainitud IP aadress *ReverseDNS* klassi, mis tagastab vastavalt IP aadressile hostinime.
- Luua hostinime päring URL-ist.

Lisaks sellele olid algsed Paberivaba Panga nõuded samuti välja kirjutatud. Edaspidises arenduse tuleks täita Paberivaba Panga *back-end* osas antud nõuded:

- Luua teenus, mis tagastaks hostnime. Kõige lihtsam oleks siinkohal luua uus täiendav API *endpoint*, mis ootab IP-aadressi *string* formaadis ning tagastab hostinime. Meetod, mis tagastab hostnime on olemas ReverseDNS klassi näol.

Front-end nõuded:

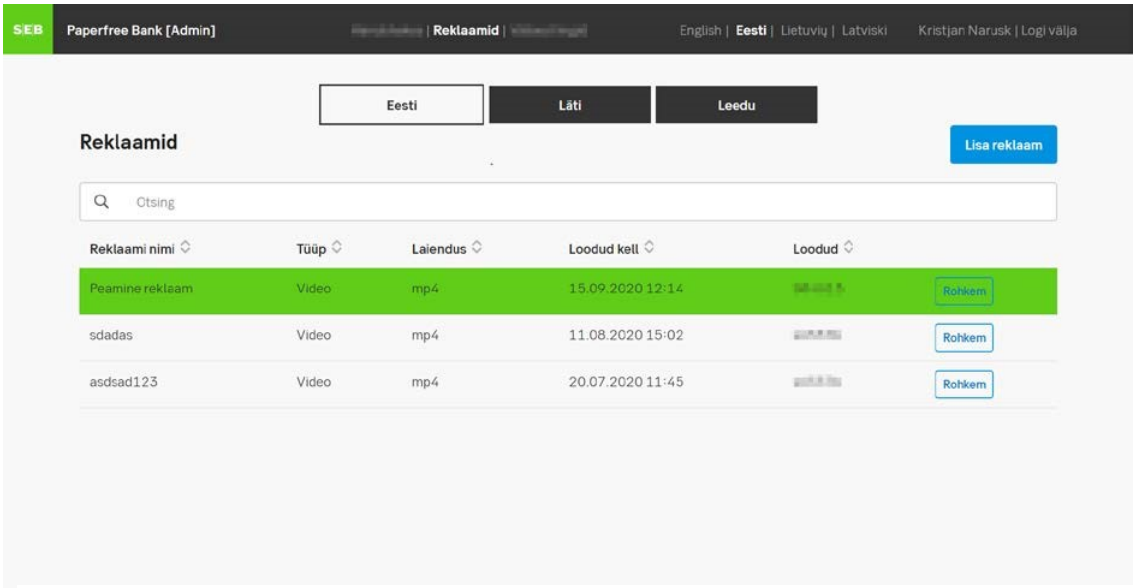
- Luua IP-aadressi päringu funktsioon, kui klient avab rakenduse brauseris. Kuna seda on keeruline luua, siis tuleks üksmeelele jõuda arhitektuuri- ning turvalisuse osakonna spetsialistidega, et milliseid lahendusi saab kasutada.

3.2 Reklaamihalduse rakendus

Paberivaba Panga rakendus on jaotatud kaheks osaks – loogika ning andmete töötlusega seotud pool ehk *back-end* ja andmete kuvamine ning graafilise liidese pool ehk *front-end*. *Front-end* oli jaotatud omakorda kaheks osaks – kliendi vaade ehk paperfree-UI ja telleri reklaamihalduse vaade ehk paperfree-admin-UI. Antud projekti arendus toimus peamiselt Paberivaba Panga *back-end,* kliendi UI ja admin UI poolel. Projekti välja töötamise käigus lähtuti puhta koodi printsiipidest, mis tulid ilmsiks nii kasutajaliidese kui ka loogika poolel. Näiteks kindlate muutujate nimetamisel kasutati neile kõige paremini sobivaid ja kirjeldatavaid nimetusi. Kui oli mõni muutuja, mille nimetamisel välja pakutavat lahendust võis mitmeti mõista, sai konsulteeritud teiste arendajatega, kus võeti vastu kõige loogilisem variant.

3.2.1 Avaleht

Avalehe puhul oli suurem osa arendusest lõpule viidud. Tehtud tööde hulgas oli „Lisa reklaam“ nupu lisamine, aktiivse reklaami esile tõstmine teistest ning riigipõhine listi sorteerimine. Listi sorteerides käivitatakse päring *back-end* poolele, mis tagastab uue JSON listi reklaami infost vastavalt riigile. Antud riikideks on Eesti, Läti ja Leedu ning pärides kasutatakse lühendeid EE, LV ja LT. Reklaamide listi päringu tulemusena esitatakse pealehel

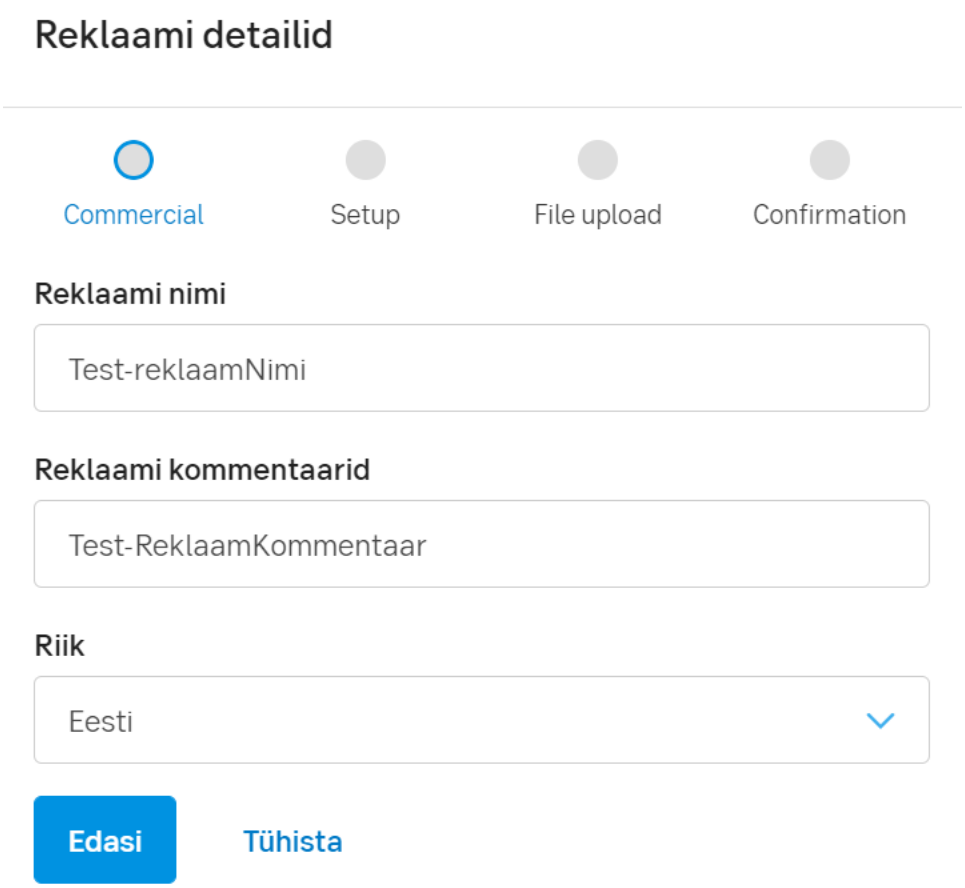


Joonis 4. Reklaamihalduse avaleht

iga reklaami kohta vastav nimi, tüüp, laiendus, loodud kuupäev ja kellaaeg ning viimaseks autori ID. Lisaks on võimalus valida iga reklaami puhul „Rohkem“, mille tulemusena avaneb valik reklaami andmeid muuta või kuvada reklaami detailid.

3.2.2 Reklaami lisamine

Reklaamihalduse põhiline mure oli uue reklaami salvestamine. Siinkohal lisati admin-UI poolele mitu uut vormi, mille tulemusena sai uue reklaami lisamisel anda reklaamile nimi, kommentaarid, riik, seadistuse aktiivsuse kuupäevalised parameetrid, seadistuse prioriteet ja kommentaarid ning lõpetuseks reklaamfail ise. Peale vormide täitmist kuvatakse sisestajale kõik eelmainitud andmed vastavalt sisestatud väärtustega. Kinnitusega rahul olles saab kasutaja reklaami salvestada või tühistada. Igal sammul on võimalus kasutajal tagasi minna eelneva vormi juurde, kasutades selleks nupp „Tagasi“. Seadistuste puhul on võimalik antud samm vahele jätta, tehes seda hiljem detailvaate alt. Siinkohal tuleb mainida ära, et salvestatud reklaami ilma seadistuseta tahvlil ei kuvata.



Joonis 5. Reklaami lisamise esimene samm

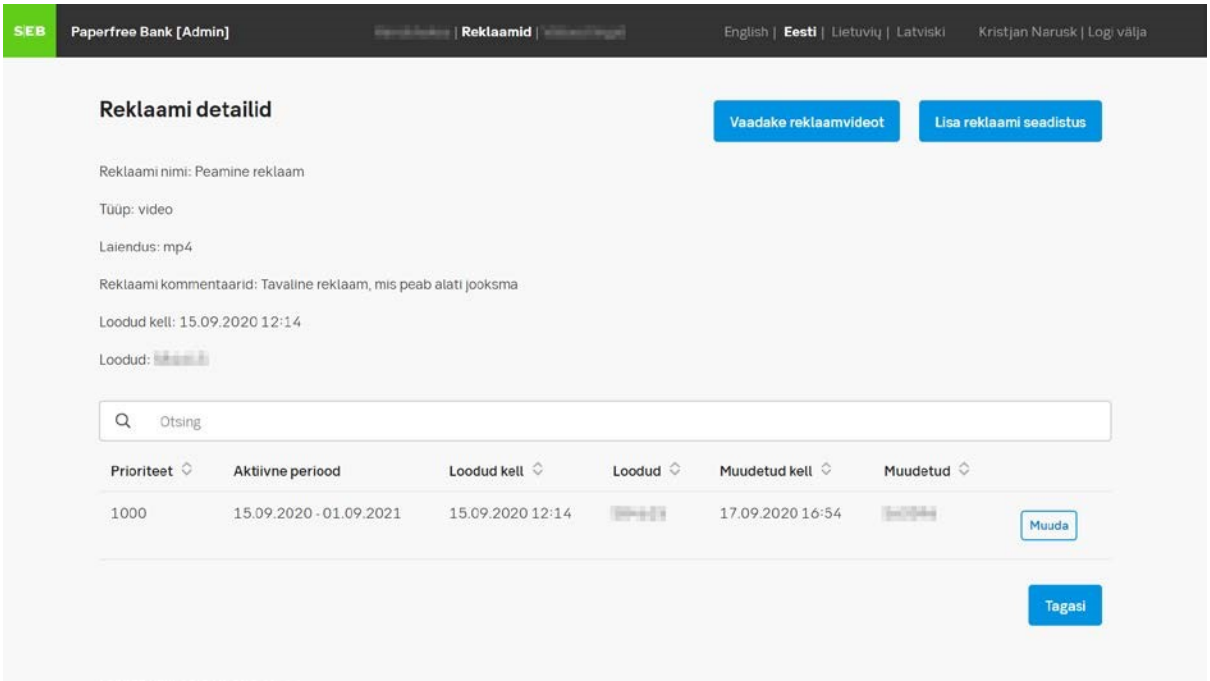
Peale kõikide vormiväljade täitmist avaneb kasutajale vaade kontrollimiseks ning sealjuures ka salvestamiseks.

Joonis 6. Reklaami salvestamise viimase sammuna kuvatav kontrollvaade.

3.2.3 Reklaami detailvaade

Reklaamihalduse jaoks oli vajadus lisamise ning avalehe täienduste kõrval ka detailvaate jaoks. Detailvaade hõlmas endas reklaamiga kaasaskäivate andmete kuvamist – avalehel oleva listis toodud andmetele on muuhulgas lisaks reklaami kommentaar ja reklaami seadistused. Seadistustel on samuti tulpadena näha selle info, nagu näiteks aktiivsuse kuupäevad, autor ja muutmise kuupäevad. Muutmine toimub seadistuste puhul samamoodi nagu reklaami endaga, vajutades seadistuse enda real asuvale nupule „Muuda“. Muutmisel on võimalik vahetada seadistuse prioriteeti, aktiivsuse kuupäevasid ja kommentaare. Seadistuste lisamine käis

sarnaselt reklaami enda lisamisega, täites ära väljad ning hiljem, peale kontrollimist, salvestades.



Joonis 7. Reklaami detailvaade koos seadistustega

3.3 Arhitektuur

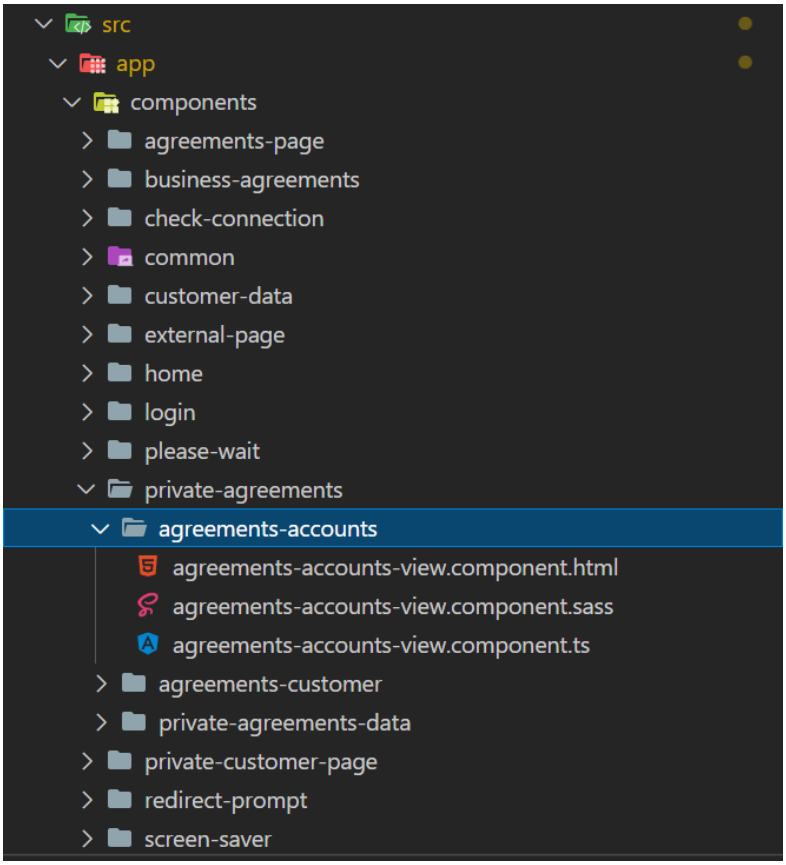
Paperivaba Panga projekti *back-end* lahendus põhines REST arhitektuuristiilil. REST arhitektuuristiili kasutatakse tavaliselt kaasaegsete veebiteenuste API-de kujundamisel. REST-i arhitektuuristiilile vastav veebi-API on REST-i API. REST API olemasolu muudab veebiteenuse *RESTful*-iks. REST API koosneb omavahel ühendatud ressurssidest. Seda ressursside komplekti nimetatakse REST API ressursimudeliks (Masse, 2011). Paberivaba API siinkohal koosnes peamiselt kontrollerist, teenusklassidest ehk *service*-itest ja mudelitest. Kontrollerid omakorda aga ei tegelenud andmete töötlemisega otse vaid suunasid saadud andmed edasi läbi määratud liideste ehk *interface*-ide implementatsiooni ehk *service* klassidele. Selle etapi eesmärgist on saavutada täielik abstraktsioon. Liidesega saame ära määrata, mida mingi kindel klassi funktsioon teeb ning millised parameetrid peavad meetodil kaasas olema. Paberivaba Panga projekti puhul on liidesed ära määratud selleks, et näidata, millised funktsioonid on andmete töötlemisel kindlalt olemas ning kasutusel. Liideste klassid on omakorda ära jagatud kontrollerite järgi – Iga kontrolleri jaoks kasutatakse eraldi liidese klassi, mis hoiab endas antud kontrolleriga seotud funktsioone. RESTful tüüpi arhitekuuri puhul on operatsioonide nagu kustuta, lisa, muuda ja hangi puhul

on kasutusel *AbstractService* liidesklass, mis hoiab baasoperatsioone, mida kasutatakse mitme kontrolleri poolt.

REST-teenuste peamine eesmärk on lahutada server ja klient. See on väga oluline, sest see aitab säilitada iga kliendirakenduse jaoks ainulaadset äriloogikat ja andmete salvestamist (Resca, 2019).

Implementatsiooni klassides on kogu töötluse tegelik loogika– siin on välja toodud funktsioonid, mis töötlevad saadud andmeid ning tagastavad vastavalt funktsiooni projekti dokumentatsioonis välja toodud parameetrid. API puhul ning Java olemasolu tõttu sai siinkohal kasutatud JavaDoc-i.

Paberivaba UI-s ehk info peegeldamise ülesandes suuri arhitektuurilisi muudatusi polnud. Kuna osad leheküljed olid juba varasemalt olemas siis sai nende põhjal jälgitud nimetamisreegleid ja jätkati uute failide ja kaustade loomisega. Siinkohal oli väga suur tähtsus eristada äri- ja erakliendi vahelisi lehekülgi ning luua vastavad mallide asukohad (Joonis 8.). Viimase ülesandena teostatud telleri *hostname* päringu puhul arhitektuurilisi muudatusi sisse viia ei olnud vajalik.



Joonis 8. Paberivaba UI arhitektuuriline jaotus

3.4 Kood

Kuna Paberivaba Panga projekti puhul on tegemist REST tüüpi API-ga, on *back-end* ja *front-end* eraldatud, mis tähendab, et mõlema puhul on kasutusel erinevad metoodikad ja programmeerimiskeeled. Andmete saatmine ühelt poolelt teisele käis teatud juhtudel kasutades selleks JSON andmevahetusvormingut. Näiteks kliendi andmete peegeldamisel NBSF-ist tuli info konverteerida JSON formaati ning seejärel saata see *ObjectNode* tüüpi parameetrina paberivaba *back-end*-i. Näide antud parameetriga Paberivaba Panga kontrollerist, mis saatis NBSF-ist tuleva kliendi info edasi paberivaba UI-le ehk *front-end-i*:

```
@PostMapping(„/{type}“)
@ApiOperation(value=“Event processing“)
@ApiResponses(value = {
        @ApiResponse(code = 200, message = „OK“),
        @ApiResponse(code = 401, message =
       „Unauthorized“)
 })
public boolean dispatchEvent (@PathVariable String type,
@RequestBody ObjectNode body) throws
PaperfreeValidationException {
```

```
        this.eventService.processEvent(type,
        body.get(„requestData).get(„hostname“).asText());
        return true;
    }
```

Joonis 9. DispatchEvent kontrolleri meetod

Back-end reklaamihalduse pool oli suuresti üles ehitatud CRUD meetoditele. Reklaamide puhul oli vaja päringutena uusi *endpoint*-e reklaami listi näol ehk oli vaja küsida andmebaasilt kõikide olemasolevate reklaamide loetuelu, üksikut reklaami ehk päringut läbi tema ID. Viimane on kasutuses näiteks detailvaate puhul. Lisaks pärimistele kuulub CRUD meetodite näol kontrollerisse ka redigeerimise, muutmise ja kustutamise meetodid. Kontrolleris eksisteerisid ainult lihtsamad meetodid, mis suunasid käskluse edasi loogika poolele.

Loogikaga tegelevate klasside ehk *service* tüüpi implementatsiooniklasside vahel olid veel liidesed ehk *interface*-id. Implemetatsiooniklassid olid viimased, mis tegelesid andmete muudatusega, olgu selleks näiteks andmebaasi käsklus, mis pidi repositooriumist kustutama mõne kindla reklaami, kasutades selleks reklaami ID-d. (Joonis 10.) Näite funktsiooni puhul on kastutusel erinevad logimisfunktsioonid, mille abil on täpne info toimunu kohta olemas. Logimisread aitavad vea juhtumisel puhul hõlpsasti üle leida, mis käskluse puhul valesti läks. Näiteks *try-catch* koodiploki puhul on selgelt aru saada, et kui *exception*-i ilmnemisel logida veakood ERROR_COMMERCIAL_CANNOT_DELETE, siis ilmneb konsooli inforida: „Error when deleting commerical. Database gave response:“ , mille järel on juba täpsem info vea kohta. Tihti võib antud veakood tekkida, kui andmebaasis ei eksisteeri parameetrina sisseantud ID-ga reklaami. Meetodis algselt päritakse reklaami objekt teise funktsiooni abil läbi tema ID. Seejärel üritatakse repositooriumist reklaam kustutada, õnnestumise korral logitakse konsooli sõnum: „*Deleted entity: “* koos reklaami infoga. Antud projektis sai lisatud päringumeetod läbi riigi parameetri ja ID numbri, reklaami ning reklaamiseadistuse kustutamis- ja redigeerimismeetodid. Varasemalt oli olemas reklaamide listi, aktiivse reklaami ja reklaamiseadistuse päringu- ning lisamismeetodid.

```
@Override
@cacheEvict(value = CacheName.COMMERCIAL, allEntries =
true)
public void delete (Long id) throws
PaperfreeValidationException {
      log.info(InfoMessages.COMMERCIAL.DELETE);
      Commercial commercial = getCommercialImpl(id);
      Try {
```

```
            commercialRepository.delete(commercial);
            log.info(InfoMessages.INFO_DELETED_ENTITY_MESSAGE +
           commercial);
        } catch (Exception e) {
            log.error(ErrorMessages.ERROR_COMMERCIAL_CANNOT_DELETE, e);
            throw new PaperfreeTechnicalException (ErrorMessages
            .Error_COMMERCIAL_CANNOT_DELETE + e.getMessage();
            PaperfreeErrorCode.CANNOT_DELETE);
           }
    }
```

Joonis 10. Reklaami kustutamismeetod implementatsioonina.

Front-end poolel oli loogika kirjas Typescript keeles. Üldjoontes sai kogu kood kirjutatud eesmärgiga vahetada ekraanipilte läbi nupuvajutuste, pärida või salvestada uut infot API jaoks. Reklaamihalduse poolel oli suureks abiks varasemate mallide olemasolu, mille põhjal sai järgida uute vormide tegemisel disaini elemente. Näiteks nuppude puhul oli võimalik kasutada samu *class* parameetreid, mille põhjal seati CSS vorming. Suures plaanis jäi disaini puhul kirjutatud koodi hulk väikseks, pigem kasutati vanu malle, mille muutujaid, siinhulgas suuruste parameetreid, vastavalt vajadusele kohendati. Antud projekti kontekstis lisati 13 privaat- ja ärikliendile mõeldud malli koos töötavate nuppudega. Viimase all on mõeldud loogikat, mis vahetab nupuvajutuse põhjal ekraani, kas edasi või tagasi, vastavalt nupule (Joonis 11.). Loogika on jaotatud kahte funktsiooni: *nextView()* ja *previousView().* Näiteks erakliendi andmete lehekülgede initsialiseerimisel väärtustatakse *currentView* muutuja ehk algne lehekülg esimeseks, INITIAL_DATA väärtuseks. Kui minnakse leheküljel edasi, väärtustatakse muutuja CONTACT_DATA väärtuseks, mille väärtust kontrollib omakorda komponendi HTML pool. Kui HTML näeb, et väärtus on muutunud, vahetatakse mall vastavalt väärtusele, siinkohal selleks oleks kliendi kontaktandmete oma. Algne, erakliendi üldine HTML mall on baasiks võetud ning sinna sisse on kirjutatud *ngIf* lausendite abil kontroll, mis seab vastavalt muutuja väärtusele aktiivseks kindla lehekülje.

```
nextView() {
 this.scrollToTop();
 switch (this.currentView() {
       case CustomerDataView.INITIAL_DATA:
             this.currentView =
             CustomerDataView.CONTACT_DATA;
             break;
```

```
            case CustomerDataView.CONTACT_DATA:
                  this.currentView =
                  CustomerDataView.ADDITIONAL_DATA;
                  break;
            case CustomerDataView.ADDITIONAL_DATA:
                  this.currentView =
                  CustomerDataView.CONFIRMATION;
                  break;
            }
      }
```

Joonis 11. Lehekülgede vahetamise loogika erakliendi andmete komponendis.

Kuna ReverseDNS ehk Chrome lehitseja laienduse eemaldamise ülesande kood sai kirjutatud NBSF *front-end* klassi nimega login.jsp, siis on see samuti antud kontekstis vajalik välja tuua. Login.jsp on üldine fail, mis hoiab endas baasloogikat ning HTML koodi, et kasutaja saaks antud süsteemi sisse logida. Kuna süsteem on mahukas ning keeruliselt üles ehitatud, on kogu sisselogimisega seotud loogika paisatud erinvate SEB Panga süsteemide vahel laiali. Probleemi lahendamisel oli esialgselt vaja leida koodijupp, mille abil kasutaja IP aadress pärida. Antud kood sai koos vanemarhitekti, Henrik Leinola abiga välja töötatud vastavalt projekti- ning turvanõuetele (Joonis 12.).

```
RTCPeerConnection = window.RTCPeerConnection || window.mozRTCPeerConnection
|| window.webkitRTCPeerConnection;

//compatibility for Firefox and chrome

var pc = new RTCPeerConnection({iceServers:[]}), noop = function(){};

pc.createDataChannel('');//create a bogus data channel

pc.createOffer(pc.setLocalDescription.bind(pc), noop);// create offer and
set local description

pc.onicecandidate = function(ice){

      if (ice && ice.candidate && ice.candidate.candidate){

            var localIP = /([0-9]{1,3}(\.[0-9]{1,3}){3}|[a-f0-9]{1,4}(:[a-
            f0-9]{1,4}){7})/.exec(ice.candidate.candidate)[1];

            pc.onicecandidate = noop;

      }

}
```

Joonis 12. Kliendi IP aadressi päringu funktsioon läbi WebRTC.

3.5 Testid

Paberivaba projekti puhul olid testimise osakaal töö tegemisel suhteliselt kõrge. See tähendab, et kõik tehtud muudatused pidid olema kinnitatud töökorras, enne kui need kliendini jõuavad. Selle jaoks lisaks koodi ülevaadetele teise arendaja poolt olid vahelülideks ühiktestid ning eraldi testija poolt tehtavad kontrolltestid.

Ühiktestide poolt kasutati Paberivaba Panga projekti *back-end*-is JUnit testimisraamistikku Java koodi testimiseks. See on lihtne ning praktiline Java klasside testimisraamistik, mis soodustab tihedat integratsiooni testimise ja arenduse vahel, võimaldades testikomplekti järk-järgult üles ehitada (Cheon, Y., & Leavens, G. T, 2002). Testide jaoks oli loodud projekti *back-end* poolele eraldi kaust, mis oli omakorda jaotatud ära erinevate klassitüüpide vahel. Projekti käigus sai kirjutatud 19 testi *CommercialController*-i ja 25 testi *CommercialService*-i jaoks.

Näite testi jaoks on välja toodud reklaami päringu testid ID kaudu, nii positiivne kui negatiivne test. (Joonis 13, 14). Joonistel on näha koodi, millel kasutatakse Mockito testimisraamistikku, et luua testimiseks *mock* andmed. Selle all on mõeldud andmeid, mis päriselt ei eksisteeri, vaid luuakse ainult testimiseks. Testimisel võetakse aluseks andmetüübid, mis on eelnevalt sätestatud, kuid reaalseid andmeid, näiteks andmebaasist, ei kasutata. Ühesõnaga kasutatakse Mockitot klasside funktsionaalsuste testimiseks, sõltumata funktsionaalsuse testimiseks andmebaasi ühendusest, atribuudifaili või failiserveri loetud funktsioonidest. Nii-öelda võltsobjektid mockivad reaalset teenusklassi. *Mock*-objekt tagastab näivandmed, mis vastavad mõnele talle edastatud võltssisendile. (Venkata, C. 2019). Antud näites proovitaks *mock* tüüpi repositooriumist leida üles suvalise testimise jaoks genereeritud ID kaudu reklaamitüüpi objekt. Sama ID kaudu üritatakse leida üles *service* klassist reklaamobjekt ning käsklusega *AssertThat* kontrollitakse kas antud objektil, mis *service* klassist leiti, on sama ID-ga. Kui on, läbib test. Peaaegu samat loogikat on kasutatud ka negatiivse testi puhul, siinkohal on ainult oodatavaks tulemuseks veakoodi tagastamine *exception*-i saamisel. Oodatava tulemuse saamisel test läbib ning saab positiivse tulemuse.

```
Test
@DisplayName("Get commercial by ID")
void whenGetCommercialsById_thenReturnCommercialRepresentation()
throws PaperfreeValidationException {
```

```
        Mockito.when(commercialRepository.findById(ID))
                .thenReturn(Optional.ofNullable(commercial));
        CommercialRepresentation commercialRepresentation
    = commercialService.get(ID);
        assertThat(commercialRepresentation.getId()).isEqualTo(ID);
        verify(commercialRepository, times(1)).findById(anyLong());
    }
```

Joonis 13. Reklaami päringu läbi ID positiivne ühiktest

```
    Test
    @DisplayName("Get commercial by ID (negative)")
    void whenGetCommercialsById_thenThrowException()
      PaperfreeValidationException exception
      = assertThrows(PaperfreeValidationException.class, () ->
        commercialService.get(ID));
      assertThat(exception.getErrorCode()).isEqualTo(PaperfreeErrorCod
      e.NOT_FOUND);
      assertThat(exception.getMessage()).isEqualTo(ErrorMessages.ERROR
      _COMMERCIAL_BY_ID_NOT_FOUND);
      verify(commercialRepository, times(1))
        .findById(anyLong());
    }
```

Joonis 14. Reklaami päringu läbi ID negatiivne ühiktest

Testimisel kasutati eraldi klassis olevaid reklaami *mock* andmeid, mis pandi testimisel reklaami muutujate väärtusteks. Näiteks olid väärtustatud ära, kõik reklaamiga kaasaskäivad muutujad: Riik, ID, failiobjekt, tüüp, kommentaarid, looja, kuupäev ja seadistuste objekt (Joonis 15.).

```
private CommercialRepresentation mockCommercialRepresentationResponse(){
    var possibleCountries = new String[]{"EE", "LV", "LT"};
    int randId = (int) (Math.random() * 100);

    String randCountry = possibleCountries[((int) (Math.random() * 3))];
    var commercialRep = new CommercialRepresentation();
    commercialRep.setCountry(randCountry);
    commercialRep.setId((long) randId);
    commercialRep.setPayload(new CommercialPayloadRepresentation());
    commercialRep.setType(CommercialType.PICTURE);
    commercialRep.setComments("test");
    commercialRep.setCreatedBy("s12345c");
    commercialRep.setCreatedAt(LocalDateTime.of(1,1,1,1,1,1));
    commercialRep.setSetups(new
    ArrayList<CommercialSetupRepresentation>());
    return commercialRep;
}
```

Joonis 15. Reklaami objekti mock andmetega väärtustamine

Projekti teises pooles tehtud arenduses teste ei kirjutatud. Lahenduse käigus valmis saadud kood oli testitav käisitsi ning *back-end* arendusega suuresti kokku ei puututud. *Front-end* koodi muudatuste jaoks käis testimine läbi arendus- ja testkeskkondade. Kui kood läbis ülevaatuse teise arendaja poolt, sai viimane liigutada läbi JIRA ülesande *branch*-is oleva koodi edasi *dev* keskkonda. Seejärel oli algarendaja ülesanne testida tehtud muudatusi selles keskkonnas ning veenduda, et kõik töötab korrapäraselt. Alles siis jõudis muudatus test keskkonda, mille järel võttis edasise ülesande käekäigu üle testija.

4 Analüüs

4.1 Tehnilise teostuse analüüs

Projekti vältel said arendajad sisendid peamiselt IT tiimijuhilt ning toote omanikult. Toote omanik oli äripoolel asuv inimene, kes suhtles omakorda suures osas AS SEB osanikega ning võtmeisikutega, presenteerides neile valmisolevaid või teostamisele minevaid ideid. Kui ideed said heakskiidu, pandi paika nõuded ning kasutusjuhud, mis said kirjutatud JIRA ülesande kirjeldusse. Kui millegi kohta tekkis küsimus või arusaamatus, sai probleem ülestoodud eraldi IT tiimijuhti ja/või toote omanikuga *standup*-il või eraldi koosolekul. Osa disainiga seotud nõudeid muutusid arendusfaasi käigus, mis tõttu tuli UI disaineril näitevisandid ümber teha ning arendajatel vajalikud muudatused sealt omakorda ellu viia.

4.1.1 Nõuded

Projekti reklaamihalduse poolel olid suures osas funktsionaalsed ja mitte funktsionaalsed nõuded valmis kirjutatud. Kvartalipõhised eesmärgid olid välja toodud SEB Panga IT jaoks mõeldud Confluence dokumentatsioonihoidla *pipeline*-is. *Confluence*-is oli eraldi projektina kirjas kogu uue Paberivaba Panga projektiga seonduv dokumentatsioon. Selle tõttu oli arendajana üsna lihtne tööd alustada, teades, millised ülesanded on esmatähtsad. Ärivajadustest lähtudes oli kriitiliselt tähtis, et reklaamfaili oleks võimalik salvestada ja et seda kuvataks paberivaba UI ekraanil. Sellest tulenevalt oli ühe nõude puhul arendajatel vaja välja selgitada, millises vormis on parim reklaamfaili salvestada. Antud probleemi puhul oli lahenduseks reklaamfail kodeerida *base64* tüüpi *string* muutujaks, kuna nii oli kõige hõlpsam salvestada ning reklaami kuvada, võttes arvesse mahtusi. Lisaks on Google Chrome brauseri sessioonmälu piiranguks 5MB, mis tähendas, et videofaili ei olnud võimailik sessioonmälus sellel kujul talletada.

Base64 kodeerimisprotsess tähistab 24-bitiseid sisendbittide rühmi 4 kodeeritud tähemärgina väljundstringidena. Vasakult paremale liikudes moodustatakse 24-bitine sisendrühm, ühendades 3 8-bitist sisendrühma. Seejärel käsitletakse neid 24 bitti 4 liidetud 6-bitise rühmana, millest igaüks tõlgitakse 64 tähe tähestikus üheks tähemärgiks (Joonis 16). Base 64 kodeering on loodud esindama suvalises järjestuses tähti kujul, mis võimaldab kasutada nii suuri- kui ka väiketähti, kuid genereeritud tekst ei pea lõpuks olema inimese jaoks loetav (Josefsson, S, 2006). Väärtus salvestati peale kodeerimist andmebaasi reklaami *payload*

muutuja alla. Reklaamfaili pärides saadeti väärtus *front-end*-i ning dekodeeriti uuesti ümber MP4 või JPEG/PNG failiks.

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

Joonis 16. Base64 kodeerimisprotsessi tähestik

4.1.2 Arhitektuur

Sarnaselt nõuetega oli ka arendaja jaoks suures osas projektiga liitudes arhitektuurilised otsused tehtud. Andmebaasi osas on kasutusel PostgreSQL andmebaasimootor, kus on

reklaamihalduse puhul jaotatud ära erinevatesse tabelitesse reklaam, reklaamseadistus ja reklaamfail ehk *payload.* Omavahelise ühendamise jaoks on kasutatud primaar- ja võõrvõtmeid. Lahendus töötab nii, et primaarvõti asub reklaamtabelis ID muutujana ning on seotud reklaamseadistuse tabelis võõrvõtme muutujaga *commercial_id*. Reklaamfaili tabeli ja reklaami tabeli vahel on ühendus loodud sarnaselt eelmisega, sidudes siinkohal faili tabeli primaarvõtme ID muutuja reklaami tabeli võõrvõtme *commercial_payload_id* muutujaga. Kuna reklaamide puhul eksisteerib tingimus, et ühel reklaamil võib olla mitu seadistust, see tähendab, et üks reklaam võib tulevikus veel korduda, on lahenduseks üks-mitmele seos. Antud lahendus parim valik sellise nõude teostamiseks.

Screen sharing ehk info peegeldamise ülesande puhul oli arhitektuuriline jaotus lahendatud täpsete kirjete ning kaustade jaotamise läbi. Siinkohal oli väga suur rõhk failide nimetamisel, et arendajatel ei tekiks tulevikus raskusi probleemi lahendamisel õigete failide leidmisega. Näiteks oli era- ja äriklientide mallid eraldi kaustades. *Service* taseme *typescirpt* koodi puhul ei pööratud suurt tähelepanu arhitektuurile – siinkohal oli suurem vajalikkus funktsioonide töötamisel ning andmete kohale jõudmisel. Paperfree-UI puhul olid funktsioonid kirjutatud üksteise alla *service* klassides vastavalt eesmärgile. Näiteks reklaamidega seotud päringud asusid *commercial-service.ts* failis.

4.1.3 Disain

Disaini puhul oli olemas kindlaks tehtud vormingud UX disaineri poolt, mis asusid sketch.com leheküljel. Sketch on disainimiseks ning prototüüpimiseks mõeldud veebirakendus, mille abil on võimalik ettevõtetel oma rakenduste visandeid interaktiivselt läbi proovida. Algselt laadis disainer leheküljele üles esimesed vormingud vajaminevatest lehtedest ning vormidest. Siinkohal oli arendajal ligipääs eelmainitud keskkonnale ning läbi Sketch.com-i oli võimalik saada täpne ülevaade vormide kõikidest parameetritest, nagu näiteks fondid ja suurused. See tegi arendaja töö palju mugavamaks – teades vajaminevaid parameetreid oli märkimisväärselt lihtsam HTML kasutades luua erinevaid tabeleid ning infokaste. Seejuures oli tähtis, et parameetrid oleks täpsed, sest antud vorme tuli arendajatel jäljendada üks-ühele, sest vormid olid välja joonistatud võttes arvesse tahvelarvuti ekraanisuurust. Disaine testides oli vajalik kas lokaalselt brauserist panna läbi arendajakonsooli paika õiged ekraanisuurused pikslites või testida otse läbi tahvelarvuti.

Eelmainitu käib nii reklaamihalduse kui ka paberivaba UI *screen sharing* ülesande kohta. Siinkohal olid hostnime päringu ülesande puhul ainsad disainimuudatused NBSF sisselogimise leheküljel. Leheküljele jalusesse lisati kasutaja lokaalse IP aadressi ning *hostname* tekstiline lausend. Lausendis olid eelmainitud parameetrid välja toodud. Antud ülesande koodimuudatused puudutasid kasutajale mitte nähtavaid komponente.

Back-end poolel mustritele suuresti rõhku ei pandud. Kõige tähtsam punkt oli arenduse jooksul, et funktsionaalsus oleks töökorras ning et kirjutatud koodi testid läbiksid.

4.1.4 Kood

Sarnaselt eelmise aasta lõputööga AS SEB Pangas teemal „Rahapesukahtlase tegevuse kohta teatise edastamise rakenduse täiendamine“ on ka selle lõputöö puhul järgitud koodi kirjutamisel peaaegu samu põhimõtteid. Näiteks on klasside nimetamisel järgitud, et iga klassi nimi algaks suure tähega ehk Upper Camel Case reegel kehtiks. Muutujate nimetamisel oli aga järgitud Lower Camel Case reeglit, mis tähendab, et fraas algaks väikese algustähega. (C.R. Reidolf, T. Tammaru, L. Väli, 2020)

Samuti on järgitud koodi kirjutamisel puhta koodi põhimõtteid, mille järgi peaks iga meetod täitma ühte eesmärki. Selle all on mõeldud, et kui näiteks Paberivaba Panga reklaamihalduse uue reklaami salvestamismeetod peab salvestama reklaami, siis ei teeks see sama kutsungi alusel midagi, mis läheks salvestamise kontekstist välja. Kui on vaja eesmärgi jaoks veel eraldi loogikat, siis see tuleb omaette funktsiooni kirjutada ning viimane algmeetodis välja kutsuda. Igat meetodit või sektsiooni eraldab üks rida ning kood on trepitud tühikutega. Loogelised avavad sulud on kirjutatud ridade lõppu ehk reavahetus toimub pärast avavat loogelist sulgu. Iga sulgev loogeline sulg on uuel real ehk reavahetus toimub enne sulgevat loogelist sulgu, v.a siis, kui sellele järgneb *else* või *else if*. (C.R. Reidolf, T. Tammaru, L. Väli, 2020)

Hostname päringu koodis on kasutatud lisaks kontrollivatele *if* lausetele lisaks *try-catch* koodiplokki, mille tulemusena *back-end* osa ei jooksuta ennast kokku ning vea puhul on eraldi loogika ette nähtud. Antud lahendust on kasutatud kolmanda sammuna, kus Chrome brauseri laienduse läbi on proovitud pärida telleri *hostname,* vea ilmnemisel on konsooli logimise käsklus koos ebasobiva *hostname*-ga (Joonis 17.). Näite koodis on vahetatud osa koodi muutujatest turvakaalutluste põhjusel välja.

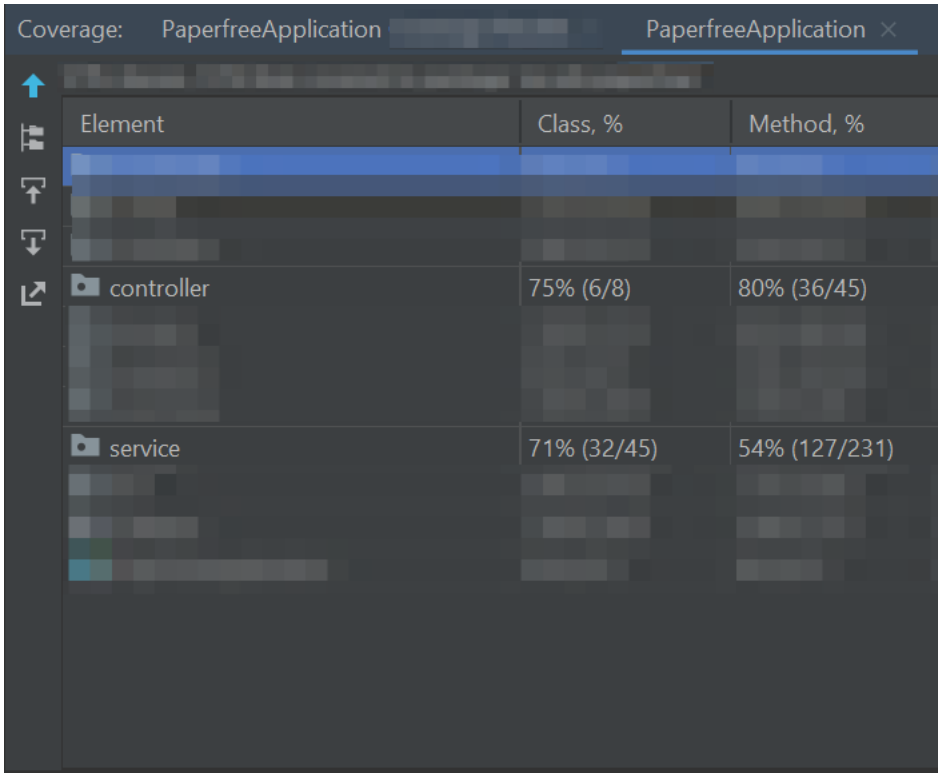
```
try {
   hostName = Chrome_Plugin("XXXXXXX-XXXX-XXXX-XXXX-
XXXXXXXXXXX");
   if(hostName == "undefined") {
      if (console) {console.warn("Computer name not found in
extension. Waiting for it.");}

      // Loop is here to be launched only in case if Computer
name was not picked up!
      for (i = 0; i < 3; i++) {
         setTimeout(function () {
            if (console){console.log("Asking computer name in
the loop", i);}
            hostName = Chrome_Plugin("XXXXXXX-XXXX-XXXX-XXXX-
XXXXXXXXXXX");
         }, 1000);

         if (hostName != "undefined") {
            break;
         }
      }

      if (hostName == "undefined") {
         throw "Undefined hostname";
      }
   }

} catch(e) {
   if (console){console.log("Hostname is undefined: " +
hostName);}
   try {
      if (console){console.error(e);}
   } catch (ex) {
      // OK
   }
}
```

Joonis 17. Try-catch koodiplokk Chrome brauseri laiendi kasutamiseks

4.1.5 Testid

Paberivaba Panga projektis oli eelnevalt sätestatud, et suur osa *back-end* arenduses tehtavad lisad peavad olema testitud ühiktestidega. Selle tõttu olid kõik reklaamihaldusega seotud kontrollerid ning *service* klassid kaetud ühiktestidega. Testimisel kirjutati testid eraldi *Tests* kaustas olevatesse klassidesse, kus oli eelnevalt ära sätestatud *mock* tüüpi andmed. Näiteks reklaami kontrolleri testimisel *mock*-iti *serivce* klassi kasutades selleks Mockito raamistiku *@MockBean* annotatsiooni. Lisaks oli *@BeforeEach* annotatsiooniga ära märgitud S*etUp* klass, millega väärtustatakse muutujad väärtustega uuesti enne igat testi. Iga kirjutatud test algas *@Test* annotatsiooniga, mille abil tundis JUnit raamistik ära, et tegu on Java testiga. Iga test sisaldas mingit sorti *Assert* lausendit, mille abil verifitseeriti oodatava ja tegeliku tulemuse õigsust.

Testimisel oli kontrollerite meetodite kaetavus 80%, klassi kaetavus 75%. *Service* klassidest oli testidega kaetud 71%, meetoditest 54% (Joonis 18.).



Joonis 18. Paberivaba Panga back-end API koodikaetavus

4.2 Kirjanduse ülevaade

Enamus kirjandusega seotud tekste oli võetud kas Google Scholar veebivaramust või Tallinna Tehnikaülikooli raamatukogu andmebaasidest. Andmebaasidest kasutati IEEE Xplore digitaalset veebivaramut ning *O'Reilly For Higher Education* IT temaatikaga andmebaasi, et leida vajalikke artikleid ning raamatuid. Antud digitaalsetest väljaannetest võetud lõike ning mõtteid sai kasutatud antud lõputööd kirjutades. Enamus kui mitte kõik koodiga seotud probleemidest ning mõttekohtadest sai lahendatud kas SEB Panga IT arendusdivisjoni kuulunud kolleegide abil või otsitud info abil läbi Google otsingumootori. Siinkohal tuleb välja tuua arendajatele mõeldud küsi-vasta stiilis StackOverflow foorumi, mille kasutamist tiimisiseseselt innustati. Eelmainitud keskkond aitas väga suurel määral koodiprobleemide ning uute lahenduste välja töötamisel kaasa.

Puhta koodi kirjutamine oli lõputöö vältel üks kindlatest kriteeriumitest lõputöö autori jaoks. Seetõttu sai küsimuste puhul viidatud raamatule Clean Code (R. C. Martin, 2008), millega oli tutvutud juba Tallinna Tehnikaülikooli "Infosüsteemide Arendamine II" aines. Raamatu abil

järgiti puhta koodi põhimõtteid just nimetamisel, nagu näiteks selgesti loetavad ning arusaadavad nimetused, väike- ja suurtähe kasutamine, funktsioonide suurused jpm. Näiteks oli suur rõhk sellel, et funktsioonid jääksid koodihulga poolest väikseks ning et nad täidaksid vaid ühte eesmärki.

4.3 Logid

Arendus algas autori jaoks 2020. aasta 3. juunil liitudes AS SEB Panga Paberivaba tiimiga. Esimese nädala jooksul sai kohanetud töökoha ning -kollektiiviga. Tiimis töötades sai ülesandeid jagatud juuniorarendaja Daniil Petuhhoviga, mentori ning juhendaja rolli kandis esimeses projekti faasis Sergei Pojev. Alljärgnevad tabelid on autori töö kohta käiv väljavõte JIRA-st Git versioonihoidlasse üleslaetud koodi kuupäevadest ning lühikirjeldustest.

10. sprint (09.06-30.06) PFB-70 ülesanne

Kuupäev	Üleslaetud koodi sõnum
11.06	Lisasin *front-end* toe reklaami salvestamiseks (pooleli)
15.06	Kuupäeva valideerimine lisatud ’Aktiivne alates’ ja ’Aktiivne kuni’ väljadele
30.06	Muutsin reklaami kommentaare ja seadistuste kommentaare tõlkefailides, reklaame lisades avaneb modaalaken järjestikuselt, TODO (tegemiseks): äriloogika arendada, et viia edasi reklaami andmed ühest modaalaknast teise.

10. sprint (09.06-30.06) PFB-88 (*back-end* arendus) ülesanne

Kuupäev	Üleslaetud koodi sõnum
10.06	Reklaami saamine ID päringu järgi, *CommericalServiceImpl commerical.get()* funktsioon ei tööta.
11.06	Lisasin 3 *endpointi:* saada reklaam ID järgi, lisada reklaamiseadistus ja uuendada reklaam (pooleli). Vaja vaadata üle Sergeiga.
15.06	Uuendamise ja kustutamise *endpoint-id* lisatud, kustutamise *endpoint* on pooleli, vaja arutada Sergei või Daniiliga, kuidas seda teha
16.06	Reklaami saamine ID ja uuendamise testid läbivad
16.06	Lisasin veateated reklaamidele
16.06	Eemaldasin tagastuslausendi reklaami kustutamise meetodilt, test ei läbi
16.06	Reklaami CRUD operatosioonide testid tehtud

16.06	Reklaamiseadistuse PUT meetod + test tehtud

11. sprint (30.06-22.07) PFB-70 Ülesanne

Kuupäev	**Üleslaetud koodi sõnum**
07.07	Reklaami kinnitus tehtud, *pipes* nüüdsest eraldi kaustas, riigi koodid on väiketähtedega + üleüldine refaktoorimine
10.07	Reklaami sisestus + uuenduse muudatused
10.07	Reklaami täpsustuste sektsioon lisatud detailide lehele
13.07	Kuupäeva formaadid
15.07	Reklaami seadistuste detailide leht, lisasin keele parameetrid reklaami kustutamiseks, väikest sorti refaktoorimine
15.07	Lisasin rippmenüü nuppude jaoks reklaami seadistuse lehele. Reklaami seadistus siiani katki – puudub seos reklaami ja reklaami seadistuse vahel
17.07	Reklaamide vormistus
20.07	Kuupäeva valideerimine + tähtsuse valideerimise kontroll lisatud reklaami seadistusele.
21.07	Lisasin EE ja LV tõlked reklaamidele ja reklaami seadistustele
21.07	Lisasin funktsionaalsuse, et seadistuste lehelt tagasi navigeerides riigi kood jäetakse sessiooni vahemällu
22.07	Muudatused reklaamidetailide komponendis, lisasin *router* paramaateri (id)
22.07	Reklaami detaili *router* parameeter lisatud

11. sprint (30.06-22.07) PFB-88 (*back-end* arendus) ülesanne

Kuupäev	Üleslaetud koodi sõnum
03.07	Lisasin LIMIT 1 *CommercialRepository findActive commerical sql* päringusse, muutsin *delete commercial mapping* POST meetodist DELETE meetodiks, lisasin *fromBody Reklaami kontrolleri salvestusmeetodisse*
03.07	Eemaldasin LIMIT, lisasin *PageRequest,* et pärida kindel aktiivne reklaam, test katki
06.07	Lisasin reklaami saamise riigi järgi kontrollerisse, lisasin *minimized mapping*-u + testid implikatsioonide ja kontrollerite jaoks
07.07	Reklaamipäring ID järgi lisatud, *commericalRepresentation* eemaldatud *SetupRepresentation* + *mapper* klassidest
10.07	Uuendamise loogika *CommercialServiceImpl* klassi jaoks
10.07	*Cachable* ajutine lahendus *CommercialServiceImpl klassis*
13.07	*getCommercialImpl* nüüdsest omab reklaami seadistuse andmeid, *StackOverflowError* eksisteerib, peab parandama.
14.07	Refaktoorimine

12. sprint (23.07-12.08) PFB-173 *Customer data (screen sharing)* ülesanne

Kuupäev	Üleslaetud koodi sõnum
05.08	Lisasin kliendi andmete lehed igaks sammuks, kliendi andmete ja kontakti andmete leht peaaegu valmis, menüüriba ja päis vajab tähelepanu
10.08	Lisavaate ja kinnitusvaate lehtede progress

11.08	Lisasin uue lehe suunamise *routing* moodulisse, lisasin *css* puuduolevatele lehtedele

13. sprint (12.08-02.09) PFB-173

Kuupäev	**Üleslaetud koodi sõnum**
18.08	Menüü ja päise loosung tehtud, võib vajada tulevikus muudatusi
21.08	Väiksed stiilimuudatused
26.08	Lisasin liidesed ekraani info ''peegeldamisest'' tulevale andmehulgale, lisasin *service* klassi kliendi andmetele
04.09	Väiksed veaparandused, stiilimuudatused kliendi andmete lehtedel

14. sprint (02.09-24.09) PFB-173 ja PFB-193 (back-end)

Kuupäev	**Üleslaetud koodi sõnum**
03.09	Lisain logimise, et püüda probleemi seoses *handshake* salvestamisega
10.09	Ärikliendi andmete komponendid (*data*)
22.09	Refaktoorimine

15. sprint (24.09-14.10) PFB-173

Kuupäev	**Üleslaetud koodi sõnum**

25.09	Rebase conflict – muudatus, et parandada teiste commitidega tekkinud viga
09.10	Lisasin üleskerimise algusesse kui navigeeerida kliendi andmete menüüdes edasi-tagasi. Lisasin uue komponendi (info-box-bottom)
14.10	Parandasin *dependencies* vead font-awesome jaoks

16. sprint (15.10-05.11) PFB-173 ja PFB-193 (back-end muudatused)

Kuupäev	**Üleslaetud koodi sõnum**
25.09	Rebase conflict – muudatus, et parandada teiste commitidega tekkinud viga
05.10	Kommenteerisin välja test lüli, sest ei läinud läbi
09.10	Valmistan ette esimest testi EventControlleri jaoks
09.10	Lisasin üleskerimise algusesse kui navigeeerida kliendi andmete menüüdes edasi-tagasi. Lisasin uue komponendi (info-box-bottom)
09.10	EventController negatiivse exceptioni test

17. sprint (05.11-26.11) PFB-237 (Update PFB back-end to the latest Java)

PFB-234 (Improvements in Admin panel) – divide commercials by country

NB! Edaspidi koodi kirjed inglese keeles.

Kuupäev	**Üleslaetud koodi sõnum**
06.11	Java version updated locally

11.11	Changes in maven file, java 14 required now
20.11	PFB-234 added selection logic, incomplete

18. sprint (26.11-16.12) PFB-234

PFB-308 (Improvements in admin-panel back-end)

Kuupäev	**Üleslaetud koodi sõnum**
27.11	Changed language dropdown items to 2-letter ISO codes, added functionality to query handshakes based on country
27.11	Changed home component to get handshakes by country
07.12	PFB-308 Changed get handshake list by country function name, wrote tests for new endpoint
07.12	PFB-308 Fixing merge
09.12	EventServiceImpl uncommented

19. sprint (16.12-06.01) PFB-234

PFB-135 ResourceServer deprecated in new Spring Boot

Kuupäev	**Üleslaetud koodi sõnum**

21.12	Added functionality for default country when handshakes
22.12	Fix for list reload

20. sprint (06.01-27.01)

Uurimuslik töö, commitid puuduvad.

21. sprint (27.01-10.02)

Kuupäev	Üleslaetud koodi sõnum
28.01	ResoruceServer left, meeting with Vladislav resulted in an acceptance that it should not be replaced/removed

02.10 – kirjutatud dokument edasiste sammude jaoks andmete kajastamiseks NBSF keskkonnast Paberivaba frondi poolele kasutades selleks WebSocketit.

22. sprint (10.02-25.02) kuni 24. sprint (03.10-23.03) oli tegu uurimusliku tegevusega ning testimisega. Esiteks kas olemasolev ReverseDNS klass töötab ning selle testimine. Teiseks tuli internetist või kolleegide vahendusel leida Javascript koodi lahendus, mist tagastaks kliendi IP aadressi. Uueks *hostname* probleemi lahenduse arenduse Git haruks sai nimeks "NBSF-8610_fixLoginJsp_Kristjan".

Kuupäev	Üleslaetud koodi sõnum
08.03	NBSF-8610 removed „BALTIC.." string from return statement
09.03	NBSF-8610 added different scenarios for getting client ip behind proxy in login.jsp

09.03	NBSF-8610 Added debug line for trunk
16.03	NBSF-8610 Added back „BALTIC...“, old nbsf trunk broke

25. sprint (24.03-06.04)

Kuupäev	**Üleslaetud koodi sõnum**
29.03	NBSF-8610 debug console log line for x forwarded for
29.03	NBSF-8610 Debug line for trunk / dev environment

26. sprint (07.04-21.04)

Kuupäev	**Üleslaetud koodi sõnum**
09.04	NBSF-8610 replaced reverseDNS (hostname query from ip) with getting hostname from URL
15.04	NBSF-8610 js webRTC solution for getting client’s IP address (ipv4)
16.04	NBSF-8610 Js ip is saved to hidden dom and java scriplet gets it from there + hostname query
16.04	NBSF-8610 changed js hidden DOM name from „data“ to „localIPadr“

16.04	NBSF-8610 Temporary solution for getting into NBSF in dev environment
16.04	NBSF-8610 Added debug lines and added js to main function, still not working
20.04	NBSF-8610 Changes by Aleksei

27. sprint (21.04-05.05)

Siinkohal on arendustegevus seoses lõputöö eesmärkidega lõppenud, järgnevates sprintides on arendustegevus toimunud väljaspool antud lõputöö skoopi ja nõudeid.

4.4 Hinnang projekti teostamise kohta

4.4.1 Projekti juhtimine ning selle teostamise protsess

Lõputöö arendamine toimus 2020. aasta 3. juunist 2021. aasta 10. maini. Selle aja vältel tegeleti suuremas osas projektiga seotud probleemide ja ülesannete lahendamisega, samas tuli ette ka tööülesandeid, mis ei olnud otseselt seotud projektiga. Lõputöö autor tegeles esimesed 3 kuud projektiga täistööajaga, järgnevad 3 kuud kuupäevast 01.09.2020 kuni 31.11.2020 tegeles autor projektiga koormusega 0,6. Seejärel, kuni 10.05.2021 on tegeletud projektiga jällegist koormusega 1,0. Koormuse vähendamine arenduse vältel on seotud koolitöö mahu kasvuga, millega tegeleti paralleelselt projekti kõrvalt.

Projekti juhendajaks oli esimese faasi vältel projekti arhitekt Sergei Pojev. Viimane oli suurepärane mentor projekti autori jaoks, andes konkreetset ning otsekohest tagasisidet ning aidates kõikide küsimuste puhul. Olukordi, mille puhul ei saanud juhendaja vastust anda, suunas viimane abi pärija edasi kolleegi juurde, kes oskas edasi abistada. Enamjaolt sai koodi ning arhitektuuriga seotud projekti küsimused siiski vastuse Sergei käest. Kuna Sergei Pojev lahkus pangast möödunud aasta septembri keskpaigast, tuli juhendajaks IT tiimi juht Aleksei Nikokošev, kes oli samuti väga hea kogemusega AS SEB Panga IT arendustiimi liige. Ainsaks kitsaskohaks tuleks välja tuua kesise suhtluse, jättes siinkohal üksteise mõistmise vahel kehva olukorda. Esines koosolekuid kus kaks osapoolt, see tähendab, et arendaja ning juhendaja said

üksteisest valesti aru ning mõtte edastamine võttis mõnel juhul kauem aega. Põhjusteks võib olla keelebarjääri tekkimine eesti ja venekeelse oskuste suhtes.

4.4.2 Üldine hinnang projektile

Üldiselt võib projekti arendusega rahule jääda. Kvartalipõhiste eesmärkidega on suudetud sammu pidada, vaatamata väiksele tiimi suurusele ning väga kogenud arhitekt-arendaja lahkumisele. Kitsaskohti leidus, mis tulid välja eriti teises faasis, näiteks *hostname* ülesande arenduse vältel. Kuna suhtlemine on mingil määral raskendatud, oli arendaja jaoks kohati ülesannete detailsus ning vajalikkus raskendatud. Osad probleemide kirjeldused olid kirjutatud välja mõningate lausevigadega, mis tegelikult muutis arendamise käigus loodavate muudatuste puhul olukordi, kus täpse mõistmise jaoks pidi probleemi detailset kirjeldust üle küsima.

Projekti vältel aitas suuresti kaasa paindliku tööaja ning töökoha võimalus. 2020 aasta juuli keskpaigas sai autor asuda tööle kodukontoris, mis COVID-19 tingitud pandeemia korral aitas suuresti kaasa arendaja vaimse tervise heaks. Seda just suuresti tänu vähenenud stressile viiruse levikuga seoses ja madalamatele transpordikuludele kodukontori võimalust arvestades.

Ülejäänud tiimiga suhtlemine sujus väga hästi, koosolekutel oli loomulikult kordi, kus energilisust ning positiivset lähenemist ehk nappis, kuid professionaalsest töökliimast tingituna said tööga seotud küsimused ning ettepanekud hõlpsasti ning sujuvalt ette kantud. Arendaja võeti samuti projektiga liitudes soojalt vastu, tutvustades esmalt kolleege ning töökohta Tornimäe kontoris, kus arendus algselt aset leidis. Pandeemiaga seotud teise laine tulekuga muutus olukord aga kiiresti – suurem osa panga töötajatest, siinhulgas ka projekti tiim suundus kodukontorisse täiskohaga tööle. Kohanemisega ei tekkinud kaebusi õnneks kellelgi, põhjuseks arvatavasti varasem kogemus keskkonna vahetamisega.

5 Kokkuvõte

Paberivaba Panga projekti lõputöö käigus sai teostatud projekti edasiarendus, mille käigus valmis reklaamihalduse paneel, suurem osa kliendi lepingute mallidest ja info peegeldamisest ning lõpuks *hostname* päringu asendamine läbi telleri IP aadressi. Paberivaba projekt on rakendus eelkõige mõeldud lihtsustamaks telleri ja kliendivahelist suhtlust ning töötamist, samas tõstes ka ettevõtte jätkusuutlikku kuvandit. Bakalaureusetöö on kirjutatud äriinfotehnoloogia õppekava tudengi poolt, kes lisanud AS SEB Panga Paberivaba Panga arendustiimi 2020. aasta juunis. Varasemalt oli Paberivaba Panga projektist olemas lahendus, mis veel tänaseni eksisteerib, kuid mille asendamiseks mõeldud rakendusega on tiim tegelenud juba 2019. aasta lõpust saadik. Vana rakendus on ajale jalgu jäämas just oma kõrgete hoolduskulude ning iganenud arhitektuuriliste lahenduste tõttu. Seetõttu võeti pangas vastu otsus arendada välja täiesti uus lahendus, kasutades selleks kaasaaegsemaid tehnoloogiaid ning arhitektuuristiile.

Lahenduste puhul oli Paberivaba Panga projekti *back-end*-is kasutatud Java programmeerimiskeelt ning Spring Boot raamistikku. *Front-end*-i jaoks oli kasutusel Typescript ning Angular JS raamistik. Hostnime ülesande arendustegevus puudutas telleritele mõeldud süsteemi NBSF, mille jaoks oli *front-end* koodimuudatused Javascriptis. Kogu dokumentatsiooni, integratsiooni, suhtluse ja ülevaate saamiseks kasutati SEB Pangas JIRAt, Confluence-i, Skype-i, Microsoft Teams-i ning Bitbucketit koos Git versioonihalduriga.

Lõputöö jooksul töötas meeskond agiilset metoodikat järgides. Arendaja ning töö autor töötas lõputöö kontekstis kokku 2020. aasta 3. juunist kuni 2021. aasta mai alguseni ning selle aja jooksul valmis suurem osa 2020. aasta 3. ja 4. kvartali ja 2021. aasta 1. kvartali oodatavatest eesmärkidest. Kõigi teostatud muudatuste ja funktsionaalsuse puhul oli lähtutud koosolekutel paika pandud nõuetest nii IT tiimijuhi kui ka toote omaniku poolt.

Kasutatud kirjandus

[1] L. Williams, A. Cockburn. (2003). *"Agile software development: it's about feedback and change,"* in *Computer*, vol. 36, no. 6, pp. 39-43, doi: 10.1109/MC.2003.1204373.

[2] Masse, M. (2011). *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. " O'Reilly Media, Inc.". (Võrgumaterjal). Available: https://books.google.ee/books?id=eABpzyTcJNIC&lpg=PR3&dq=REST api&lr&pg=PR3 - v=onepage&q&f=false
[Kasutatud 02.05.2021]

[3] Resca, S. (2019). *Hands-On RESTful Web Services with ASP.NET Core 3.* " O'Reilly Media, Inc.". (Võrgumaterjal). Available:

[Kasutatud 02.05.2021]

[4] Cheon, Y., & Leavens, G. T. (2002, June). *A simple and practical approach to unit testing: The JML and JUnit way*. In European Conference on Object-Oriented Programming (pp. 231-255). Springer, Berlin, Heidelberg. (Võrgumaterjal). Available: https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=1337&context=cs_techreports
[Kasutatud 08.05.2021]

[5] Venkata, C. (2019). *Usage of Mockito.* (Võrgumaterjal) Available: https://medium.com/art-of-coding/usage-of-mockito-674db441a334
[Kasutatud 08.05.2021]

[6] Josefsson, S. (2006). *The base16, base32, and base64 data encodings* (pp. 1-18). RFC 4648, October. (Võrgumaterjal) Available: https://www.hjp.at/doc/rfc/rfc4648.html
[Kasutatud 09.05.2021]

[7] C.R. Reidolf, T. Tammaru, L. Väli. (2020). *AS SEB Pank Eesti rahapesukahtlase tegevuse kohta teatise edastamise rakenduse täiendamine.* [Kasutatud 09.05.2021]

[8] R. C. Martin. (2008) *Clean Code: A Handbook of Agile Software Craftsmanship*, New Jersey: Prentice Hall

Lisa 1 - Eneseanalüüs

Projekti arendamine algas minu jaoks sissejuhatusega SEB Panga tööpoliitikasse. Läbida tuli kohustuslikus korras kõigile uutele liikmetele mõeldud koolitused. Nende järel tutvustati mind projekti tiimiga ning anti ülevaade panga kontorist. Seejärel seadistasin koos algse juhendaja, Sergei Pojeviga oma tööprogrammid, mida arendamiseks kasutama hakkasin. Siinkohal tekkisid algselt mõningad tõrked, kuna programmide versioonid olid kõige uuemad ning osade programmide ligipääsu tellimisega läks aega.

Esimese faasi arendamine sujus minu jaoks väga edukalt ning kindlasti arendavalt. Selle perioodi jooksul tundsin, et õppisin tarkvaraarendusest kõige rohkem. Seda just sellepärast, et kogu aeg oli mul väga kogenud mentor ja juhendaja kõrval, kelle poole sain küsimuste puhul pöörduda. Õppimisega seoses sain kogemust kõige paremini tehes ise vigu ning proovides erinevaid lahendusi läbi, õnneks ei teinud ükski kolleeg minu tööd minu eest ära vaid pigem andis juhtnööre ja abistavaid näpunäiteid, kui ma millegagi väga hätta jäin. Tore oli näha ja kinnitust saada sellele, et see mida koolis õppisin, oli täielikult kasutusel ka ettevõttes – näiteks REST raamistik, CRUD meetodid ja osad *front-end* arendusega seotud märksõnad nagu HTML ja CSS. Hea oli see, et tänu läbitud ainetele ei olnud reaalne tarkvaraline maailm võõras ning hirmuäratav vaid pigem väljakutsuv ja keskendudes vägagi tehtav.

Loomulikult tuli ette olukordi, kus motivatsioon langes ning arendus aeglustus. Sellised hetked leidsid aset tavaliselt pikkade koosolekute ajal või peale neid, nagu näiteks sprindi planeerimised. Sellised koosolekud olid üsna tüütavad ning pikalevenivad, kuna tihti oli vaja pikalt kuulata teemasid, mis ennast ei puudutanud. Lisaks langes motivatsioon siis, kui mõnda arendatavat komponenti või funktsionaalsust ei saanud juba pikemat aega tööle. Loomulikult tuleb ette olukordi, kus esimese proovimisega lahendus ei kanna vilja, kuid eriti väsitavaks teevad arendustegevuse olukorrad, kus ühe väikse probleemi lahendamiseks kulub vahel mitu päeva.

Teise faasi jooksul algas arendus positiivselt – selge olid nõudmised ja ülesanded, mida oli vaja teha ning esialgu tundus, et kogu tegevuslik kava ei jää ajale jalgu. Mallide tegemine sujus hästi, st. problemaatilisi kohti oli võrdlemisi vähe ning ülesanded said vastavalt *story point* süsteemile lahendatud. Olukord halvenes märgatavalt kui, septembris lahkus arhitekt ning juhendaja Sergei Pojev, kelle abile enam lootma ei saanud jääda. Siinkohal aitas küsimustega Aleksei Nikokošev või Daniil Petuhhov, mõlemad arendusega seotud inimesed.

Kahjuks olid tööd juba kuhjunud päris palju ka nende jaoks peale, mis tõttu abistamisele jäi vähe aega ning probleemidega tuli ise toime tulla. Selle tõttu langes ka töö kiirus ning JIRA ülesandeid tuli tihti edasi kanda tulevastesse sprintidesse.

Teise faasi jooksul, täpsemalt hostnime ja brauseri liidese eemaldamise ülesande puhul häiris isiklikult kõige rohkem uurimuslik osa, mille jooksul tundsin, et ei ole tiimi jaoks enam kasulik. Sellised tunded tekkisid tänu vähesele või osati puudulikule koodikirjutamisele, kuna tegeleda tuli enda jaoks küsimuste esitamisega nagu: „Miks see ei tööta?“ või „Miks seda varem ei ole kasutusele võetud?“. Nendele küsimustele tuli vastused leida ning läbi vastuste panna kirja arenduslik kava.

Kuigi esines mitmeid probleeme nii motivatsiooniga kui ka aja leidmisele, leian, et antud projekt on siiski tulus nii ettevõttele kui ka minule kui tudengile. Enda hinnangu järgi sain kogemuste suhtes rohkesti juurde ja õppetunnid, mille arengu jooksul sain, mööda selga maha ei jooksnud. Selle tõttu olen tohutult tänulik ülikoolile ning ettevõttele, et selline võimalus reaalset kogemust saada on tudengitele võimaldatud.

Lisa 2 – Logide kirjeldus ning tööaeg

Arendustöö vältel jagasin projekti tööülesandeid AS SEB Panga Paberivaba projekti meeskonda kuuluva juuniorarendaja Daniil Petuhhoviga. Projekti vältel jaotati ülesanded üksteise vahel sobivuse alusel – algselt, st. reklaamihalduse puhul jaotas projektijuht kogemuse järgi, hiljem juba said arendajad valida huvipakkuvaid ning rohkem kõnetavaid ülesandeid ise. Reklaamihalduse puhul töötasin suures osas Paberivaba Panga *back-end* testide ning kontrolleritega, lisaks kirjutasin välja reklaami lisamise vormide HTML mallid, vormide vahetamiseloogika, üksiku reklaami detailvaate malli. Lisaks töötasin välja menüüvaliku, mille järgi reklaame sorteerida – selle jaoks jällegist oli vaja uus *endpoint* lisada *back-end-i,* kirjutada implementatsiooniloogika ning ühiktestid.

Paberivaba UI puhul arendasin välja kõik 13 siiani lisatud kliendi andmete malli, mille puhul sai kirjutatud eraldi HTML ja CSS kood. Samuti päise- ning jaluse failid, lehekülgede vaheline navigeerimine ja *interface* ehk liidesfailid äri- ja privaatkliendi andmete jaoks. Siinkohal arendas Daniil Petuhhov välja andmete saatmise NBSF-ist Paberivaba *back-end*-i ning omakorda viimasest *front-end* poolele. *Front-end*-is kirjutas Daniil välja samuti loogika, et info jõuaks mallidele ning õigetesse lahtritesse.

Hostname ülesande puhul on kogu uurimuslik ja arendusega seotud töö tehtud minu poolt.

Antud bakalaureusetöö koosneb Äriinfotehnoloogia eriala aines „Meeskonna projekt: tellimus“ tehtud töö ning lõputöö tulemusel. Meeskonnaprojekti raames sai töötatud kokku ca. 440h, millest on maha arvutatud 5 puudutud päeva ehk 40 töötundi. Lõputöö raames, st. 2020. aasta septembrist 2021. aasta mai alguseni on kokku töötatud ca. 1192h, millest on maha võetud 20 puudutud päeva ehk 160 töötundi. Töötundide arvutamisel on välja jäetud tähtpäevad ning riigipühad.

Lisa 3 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Kristjan Narusk

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „AS SEB Pank Paberivaba projekti edasiarendus“, mille juhendaja on Tõnn Talpsepp.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

17.05.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.